tokopedia

Finding Balance

Performance Optimization vs Code Maintainability with Golang Reflection

Muhammad Auliya

Technical Architect Tokopedia
(Merchant Acquisition)

linkedin.com/in/gusaul
github.com/gusaul

Apr 2016
Software Engineer

Jul 2017
Senior Software Engineer

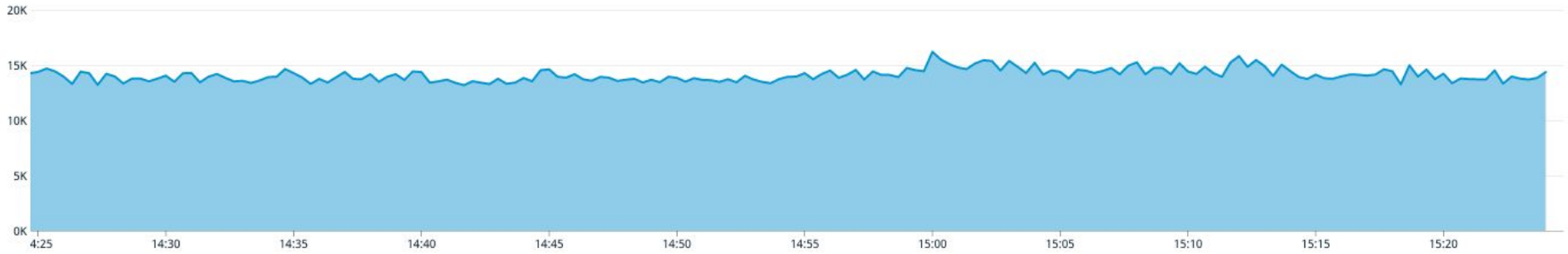
Jan 2018
Software Engineer Lead

Aug 2019
Technical Architect

Background Story

The Fact:

- Get Product Data Endpoint
- Around **15k Request per second** for BAU (**multiple times** at peak event time)



Background Story

The Fact about Get Product Data:

- Has around 20 group attributes
  - Basic Data, Inventory, Picture, Wholesale, Stats, Brand, Flags, Tags, etc ….

- Ability to get multiple data in one request
  - Become **two dimensional** data, product IDs *axis*, and Attribute *axis*

Background Story

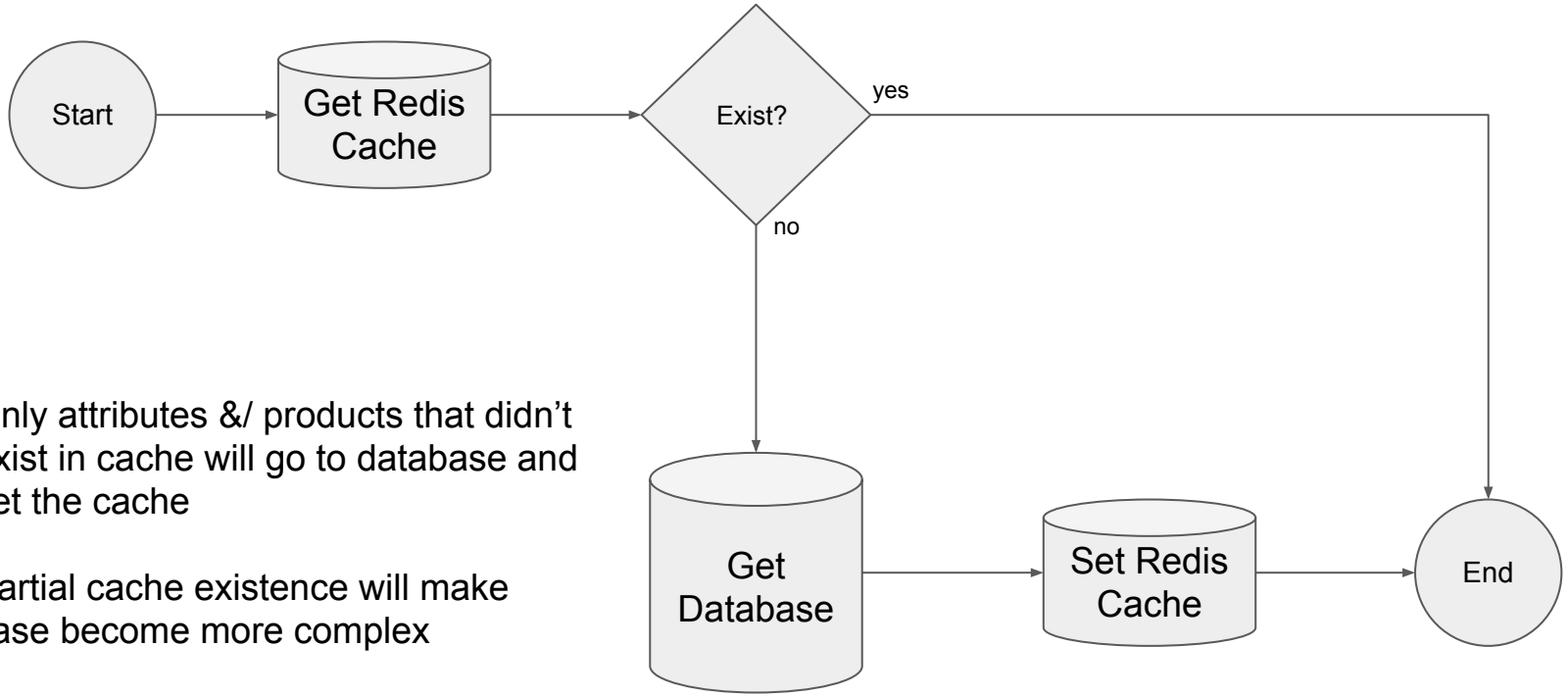
For performance reason:

- Every request can get only attributes they needs.
- Cache every request for certain period of time

Background Story

It become complex because those 2 dimensional request + caching flow

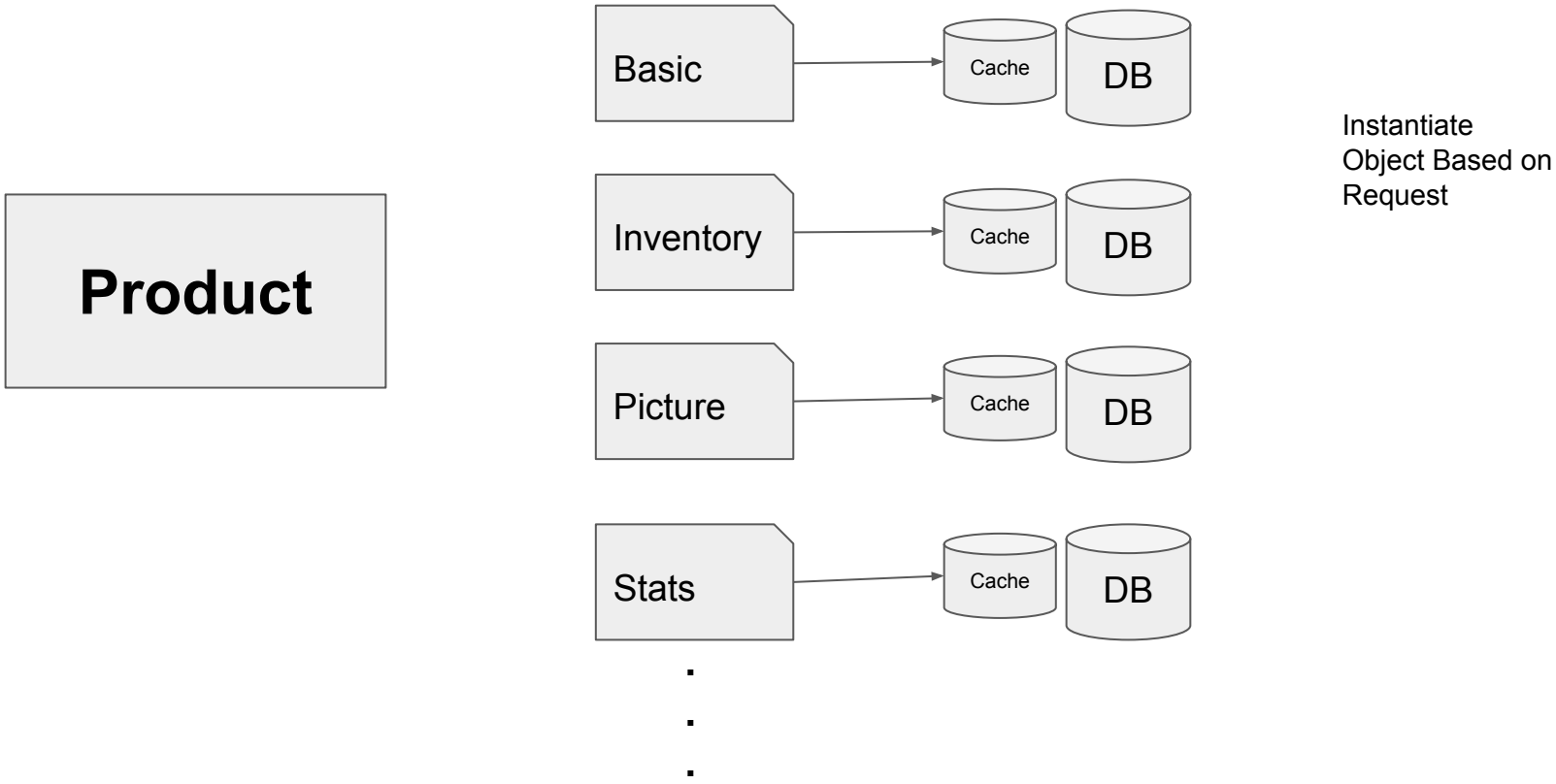
Get Resource Flow



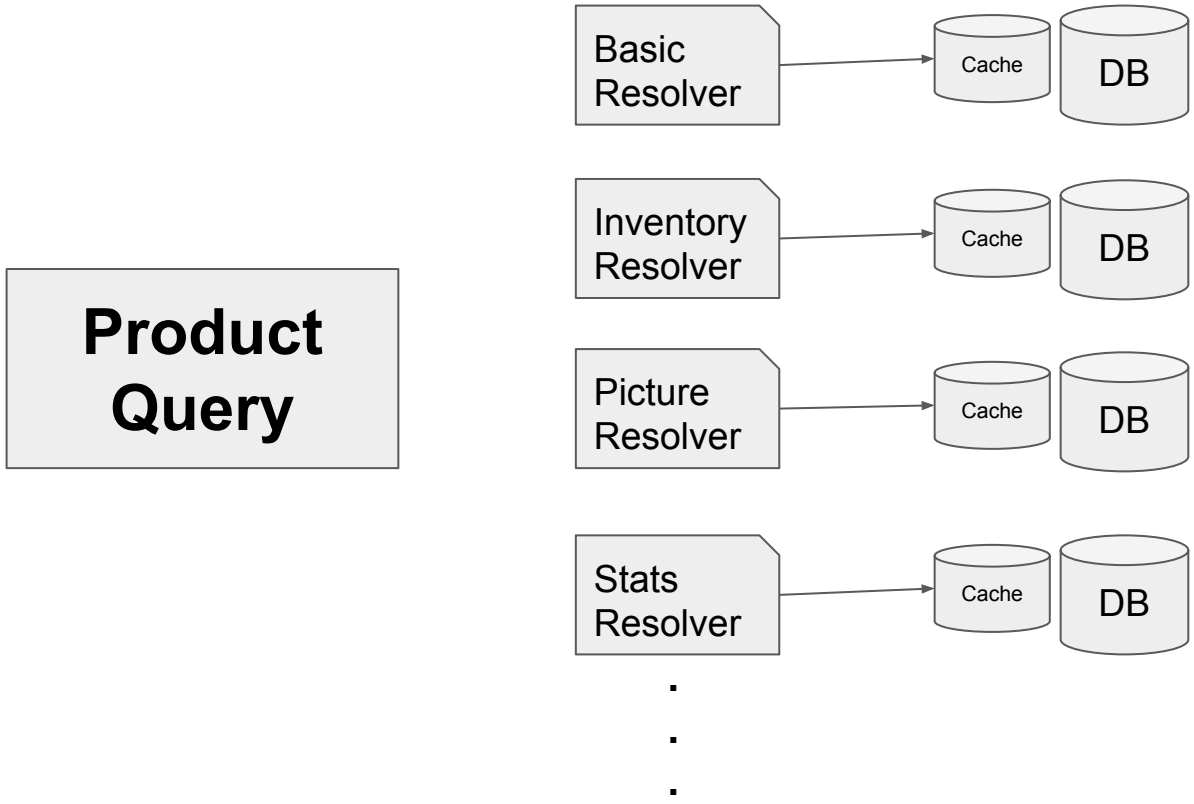
Only attributes &/ products that didn't exist in cache will go to database and set the cache

Partial cache existence will make case become more complex

Common Solution (Object Oriented Approach)



Common Solution (GraphQL Approach)



Drawbacks

- **Performance Issue**
  - Multiple get resource roundtrip (**15k RPS** leads to **700k QPS** in redis)
  - **Fact:** all attributes cache stored in same redis instances.
    all attributes stored in same database.
  - **Fact:** 80% - 90% traffic goes to redis cache.
  - All redis cache stored in single hash key
  - Utilize **Redis Pipeline**.

- **Repetitive Code Flow**
  - Expectation behavior for every object must be same.
  - DRY

Code Maintainability vs Performance Optimization

- Approaches that commonly used for code maintainability can lead us to performance issue
- But make it to work towards performance optimization without good code structure, highly possible to lead us to less maintainable code

How to achieve both?

Proposed Solution

- **Generic Flow**
  - Single flow, adaptable for all kind of struct/object.
  - Implement common interface
  - Open for Extension, closed for Modification

```
type Resource interface {
     GetCacheKey() string
     GetCacheField() []string
     ApplyCache() bool
     GetQuery() string
     GetIdentifier() int64
}
```

- **Prerequisite**
  - Has ability to transform value from Redis/DB to every struct in a generic way
    - **SQL database** -> `StructScan` from *github.com/jmoiron/sqlx*
    - **Redis** -> Create our own scanner with **reflect** package
      - Single field
      - String Array
      - JSON/Array of JSON

Reflect Package

“Package reflect implements run-time reflection, allowing a program to manipulate objects with arbitrary types. The typical use is to take a value with static type interface{} and extract its dynamic type information by calling TypeOf, which returns a Type.

A call to ValueOf returns a Value representing the run-time data. Zero takes a Type and returns a Value representing a zero value for that type.”

- **TypeOf**
  - check datatype, find field by name/index, tag, find method, etc.
- **ValueOf**
  - check value, find method, invoke method, etc.

Details reflect: ***golang.org/pkg/reflect***
Deep Understanding: ***golang.org/doc/articles/laws_of_reflection.html***

Struct Tag Setter

```
type Basic struct {
    ProductID       int64  `db:"product_id" cache:"product_id" setter:"SetProductID"`
    Name            string `db:"product_name" cache:"name" setter:"SetName"`
    Description     string `db:"desc" cache:"desc" setter:"SetShortDesc"`
}

func (b *Basic) SetProductID(value string) (err error) {
    b.ProductID, err = strconv.ParseInt(value, 10, 64)
    return err
}

func (b *Basic) SetName(value string) error {
    b.Name = value
    return nil
}

func (b *Basic) SetShortDesc(value string) error {
    b.ShortDesc = value
    return nil
}
```

Struct Tag Setter

```
type Pictures struct {
    Data []Picture `cache:"product_pictures" setter:"SetPictures"`
}

type Picture struct {
    PictureID int64  `db:"picture_id" json:"picture_id"`
    FilePath  string `db:"file_path" json:"file_path"`
    FileName  string `db:"file_name" json:"file_name"`
}

func (p *Pictures) SetPictures(value string) (err error) {
    return json.Unmarshal([]byte(value), &p.Data)
}
```

Apply Cache to Struct

```go
func (c *CacheProp) Apply(dest interface{}, data map[string]string) (isCompleted bool) {

    // get reflection type and value from struct
    objType := reflect.TypeOf(dest)
    objVals := reflect.ValueOf(dest)

    for i := 0; i < objType.Elem().NumField(); i++ {
        // get struct field attribute
        cacheField := objType.Elem().Field(i).Tag.Get(cacheTag)
        setterName := objType.Elem().Field(i).Tag.Get(setterTag)

        //  method setter must has only 1 string arg and 1 return field
        setterMethod := objVals.MethodByName(setterName)
        if setterMethod.Type().NumIn() != 1 || setterMethod.Type().In(0).Kind() != reflect.String
            || setterMethod.Type().NumOut() != 1 {
            continue
        }

        // get value from cache map
        cacheVal, cacheExist := data[cacheField]
        if !cacheExist {
            continue
        }

        // invoke setter method
        result := setterMethod.Call([]reflect.Value{reflect.ValueOf(cacheVal)})
        if len(result) > 0 && result[0].IsNil() {
            c.applied++
        }

    }

    return c.applied == len(c.Fields)
}
```

Generic Main Flow

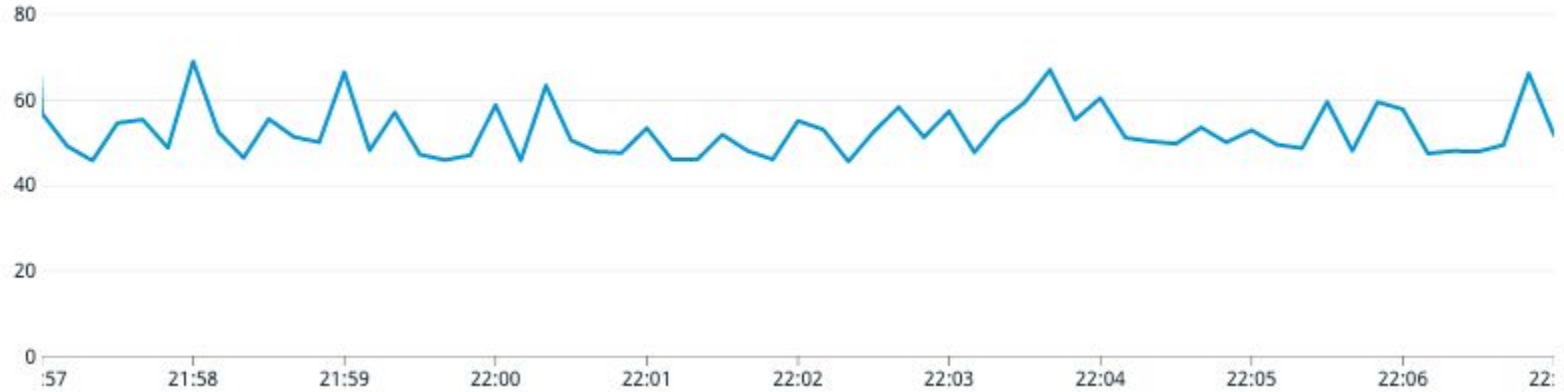
```
Input: productIDs []int64, o Options
Process:
      getters := RegisterGetter(o)
      for pid in productIDs:
            for g in getters:
                  objGet[pid] = g.New(pid)
                  cacheKeys[g.GetCacheKey()].append(g.GetCacheFields())
            end
      end
      cacheResult := getRedisCache(cacheKeys)
      for pid in productIDs:
            for og in objGet[pid]:
                  isCompleted := og.ApplyCache(og, cacheResult[pid])
                  if !isCompleted {
                        queryGroup.append(og)
                  }
            end
      end
      if len(queryGroup) > 0 {
            getFromDatabase(queryGroup)
            setCache(queryGroup)
      }
      return objGet
```

Impact

- 18 Product IDs
- Get All Attributes
- ~ 800 RPS
- Result : Max Latency 250ms - 1,5Kms

Impact

- 18 Product IDs
- Get All Attributes
- ~ 800 RPS
- Result : Max Latency 45ms - 65ms (**reduced 80%**)



DEMO

Full Code Example

github.com/gusaul/go-jakarta


Thank you!
tokopedia.com/about