How about Datastore

GCPUG Shonan vol.16 feat.Datastore

https://gcpug.jp

GCPUG
Google Cloud Platform User Group

検証プログラムのデプロイ

Case1 トランザクション同時実行の制約
Case2 トランザクション内、Entity Groupの制約
Case3 トランザクションの有効期限
Case4 task queueはroll back対象？

検証プログラムのデプロイ

```
# in cloud shell
$ git clone https://github.com/saizz/how-about-datastore
$ cd how-about-datastore
$ export GOPATH=$(pwd)

# change application in app.yaml
$ vi src/backend/app.yaml

# deploy
$ gcloud app create --region=asia-northeast1 --project=xxx
$ goapp deploy src/backend
```

Case1 トランザクション同時実行の制約

https://cloud.google.com/appengine/docs/standard/go/datastore/transactions

Using transactions

A *transaction* is a set of Cloud Datastore operations on one or more entities. Each transaction is guaranteed to be atomic, which means that transactions are never partially applied. Either all of the operations in the transaction are applied, or none of them are applied. Transactions have a maximum duration of 60 seconds with a 10 second idle expiration time after 30 seconds.

An operation might fail when:

- Too many concurrent modifications are attempted on the same entity group.
- The transaction exceeds a resource limit.
- Cloud Datastore encounters an internal error.

In all these cases, the Cloud Datastore API returns an error.

Case1 トランザクション同時実行の制約

https://cloud.google.com/appengine/docs/standard/go/datastore/entities

As mentioned above, an entity group is a set of entities connected through ancestry to a common root element. The organization of data into entity groups can limit what transactions can be performed:

- All the data accessed by a transaction must be contained in at most 25 entity groups.
- If you want to use queries within a transaction, your data must be organized into entity groups in such a way that you can specify ancestor filters that will match the right data.
- There is a write throughput limit of about one transaction per second within a single entity group. This limitation exists because Cloud Datastore performs masterless, synchronous replication of each entity group over a wide geographic area to provide high reliability and fault tolerance.

Case1 トランザクション同時実行の制約

https://cloud.google.com/appengine/docs/standard/go/datastore/structuring_for_strong_consistency

Structuring Data for Strong Consistency

フィードバックを送信

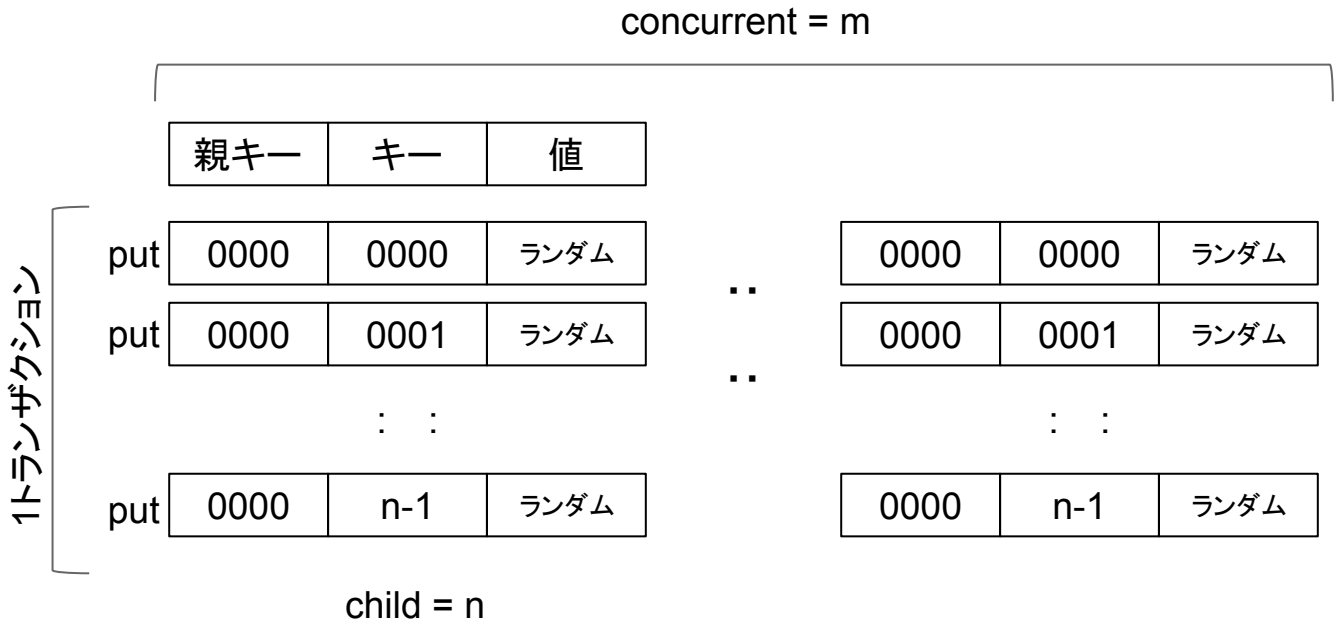
Google Cloud Datastore provides high availability, scalability and durability by distributing data over many machines and using masterless, synchronous replication over a wide geographic area. However, there is a tradeoff in this design, which is that the write throughput for any single *entity group* is limited to about one commit per second, and there are limitations on queries or transactions that span multiple entity groups. This page describes these limitations in more

Case1 トランザクション同時実行の制約

大事なことなので（ry

Case1 トランザクション同時実行の制約

同じEntity Groupに対して・・・



Case1 トランザクション同時実行の制約

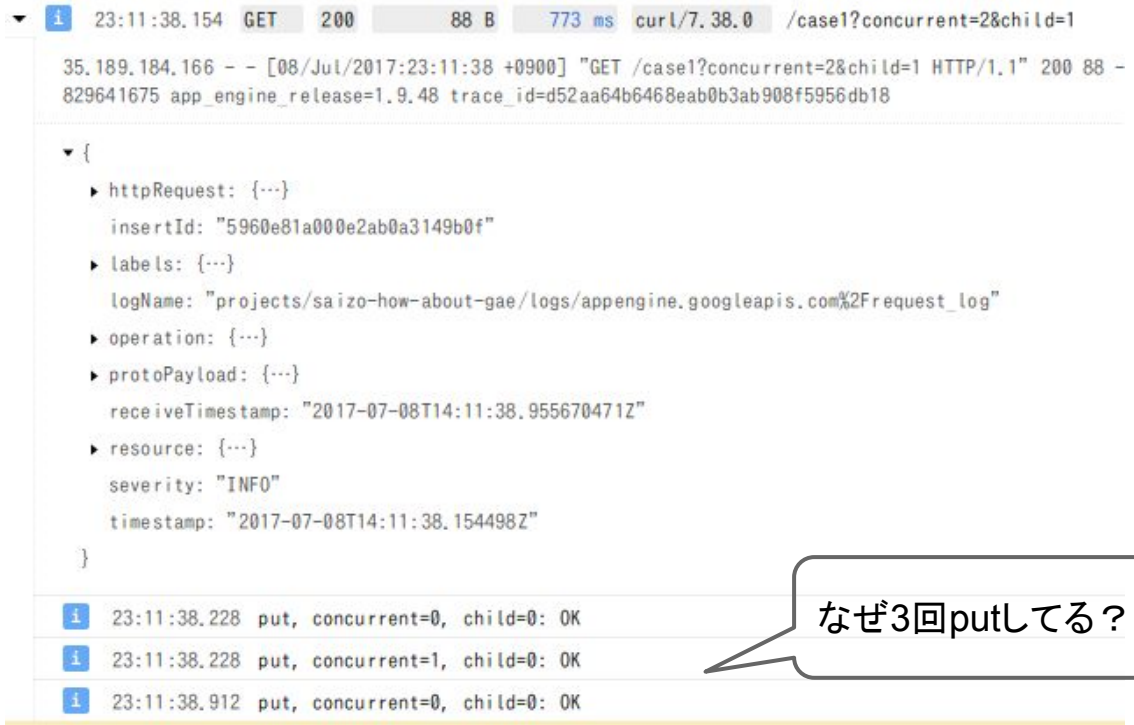
同じEntity Groupに対して・・・

concurrent / child	1	2	3	10	20
1	○				
10	○				
50	○				
100	○				

1tx(concurrent=1)内なら1秒間に何回oprationしても問題ない。
それを、gorutineで同時に行ったらどうなるか

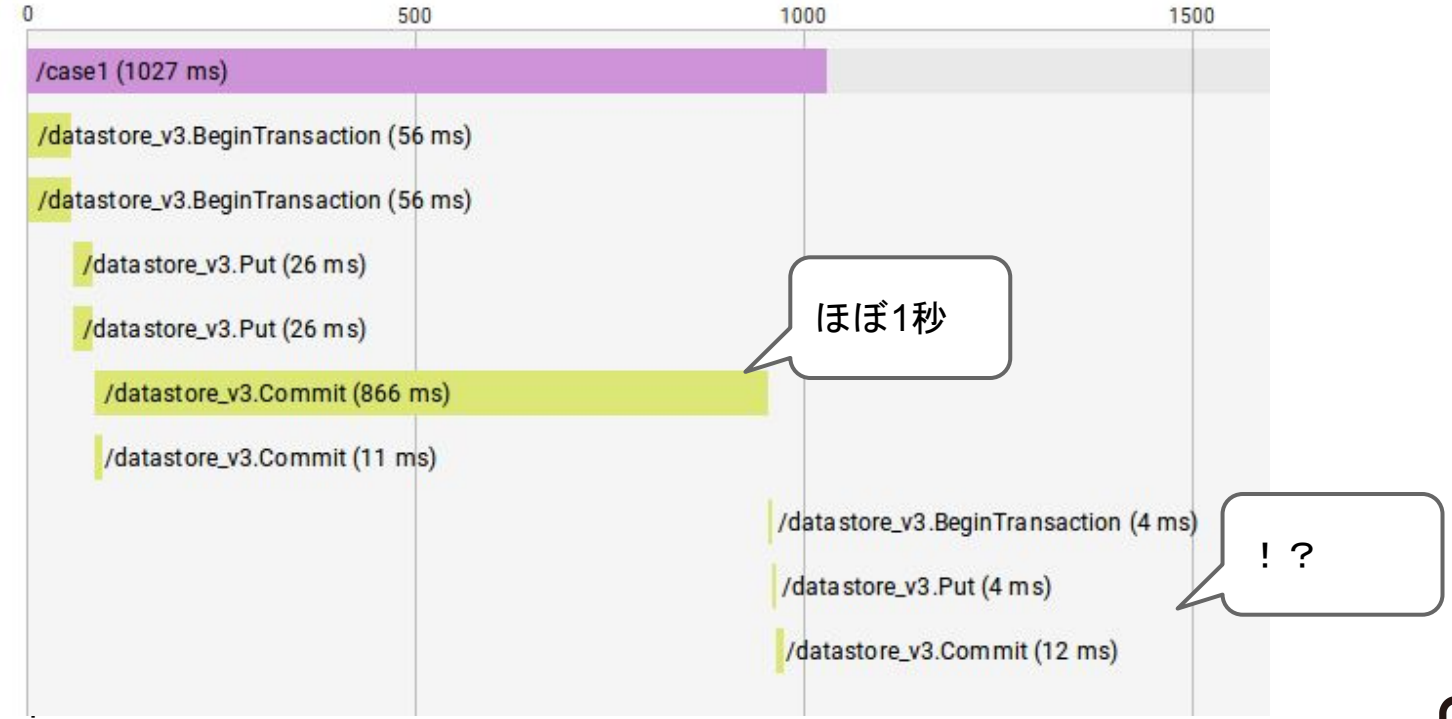
Case1 トランザクション同時実行の制約

https://xxx.appspot.com/case1?concurrent=2&child1



Case1 トランザクション同時実行の制約

https://xxx.appspot.com/case1?concurrent=2&child1



Case1 トランザクション同時実行の制約

お前の仕業か

```
func RunInTransaction(c context.Context, f func(tc context.Context) error, opts *TransactionOptions)
error {
    xg := false
    if opts != nil {
        xg = opts.XG
    }
    attempts := 3
    if opts != nil && opts.Attempts > 0 {
        attempts = opts.Attempts
    }
    for i := 0; i < attempts; i++ {
        if err := internal.RunTransactionOnce(c, f, xg); err != internal.ErrConcurrentTransaction {
            return err
        }
    }
    return ErrConcurrentTransaction
}
```

Case1 トランザクション同時実行の制約

https://cloud.google.com/appengine/docs/standard/go/datastore/transactions

When two or more transactions simultaneously attempt to modify entities in one or more common entity groups, only the first transaction to commit its changes can succeed; all the others will fail on commit. Because of this design, using entity groups limits the number of concurrent writes you can do on any entity in the groups. When a transaction starts, Google Cloud Datastore uses optimistic concurrency control by checking the last update time for the entity groups used in the transaction. Upon commiting a transaction for the entity groups, Google Cloud Datastore again checks he last update time for the entity groups used in the transaction. If it has hanged since the initial check, an error is returned. For an explanation of entity groups, see the Cloud Datastore Overview page.

Case1 トランザクション同時実行の制約

https://cloud.google.com/appengine/docs/standard/go/datastore/reference

```
var (
    // ErrInvalidEntityType is returned when functions like Get or Next are
    // passed a dst or src argument of invalid type.
    ErrInvalidEntityType = errors.New("datastore: invalid entity type")
    // ErrInvalidKey is returned when an invalid key is presented.
    ErrInvalidKey = errors.New("datastore: invalid key")
    // ErrNoSuchEntity is returned when no entity was found for a given key.
    ErrNoSuchEntity = errors.New("datastore: no such entity")
)
```

```
var Done = errors.New("datastore: query has no more results")
```

Done is returned when a query iteration has completed.

```
var ErrConcurrentTransaction = errors.New("datastore: concurrent transaction")
```

ErrConcurrentTransaction is returned when a transaction is rolled back due to a conflict with a concurrent transaction.

Case1 トランザクション同時実行の制約

結果は

concurrent / child	1	2	3	10	20
1	○	△	△	△	△
10	○	△	△	△	×
50	○	△	△	×	×
100	○	△	△	×	×

○	成功
△	成功（リトライ）
×	失敗

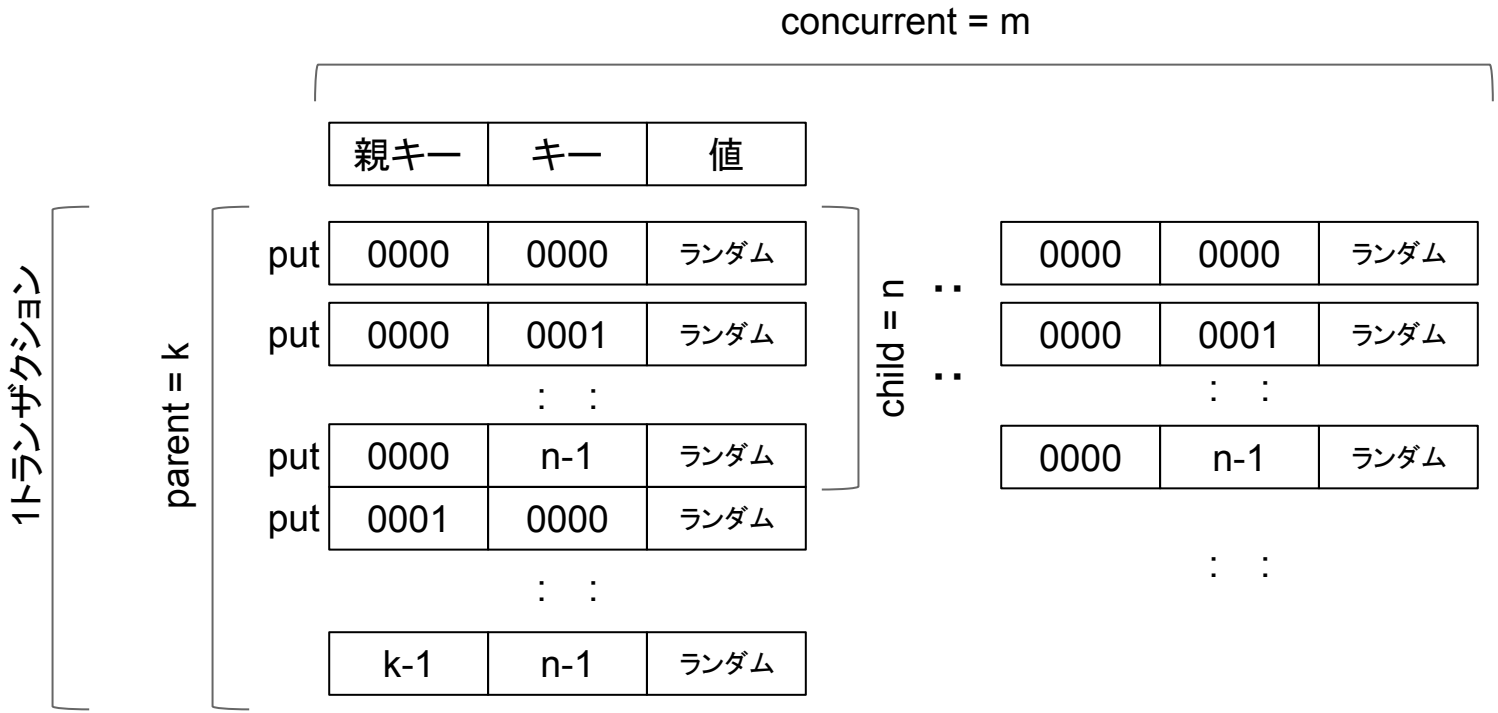
Case2 トランザクション内、Entity Groupの制約

https://cloud.google.com/appengine/docs/standard/go/datastore/entities

As mentioned above, an entity group is a set of entities connected through ancestry to a common root element. The organization of data into entity groups can limit what transactions can be performed:

- All the data accessed by a transaction must be contained in at most 25 entity groups.
- If you want to use queries within a transaction, your data must be organized into entity groups in such a way that you can specify ancestor filters that will match the right data.
- There is a write throughput limit of about one transaction per second within a single entity group. This limitation exists because Cloud Datastore performs masterless, synchronous replication of each entity group over a wide geographic area to provide high reliability and fault tolerance.

Case2 トランザクション内、Entity Groupの制約



Case2 トランザクション内、Entity Groupの制約

concurrent = 1

child \ parent	1	2	25	26	30
1	○				
10	○				
50	○				
100	○				

parent = 1ならCase1と変わらない。2以上の場合にどうなるか。

Case2 トランザクション同時実行の制約

https://xxx.appspot.com/case2?concurrent=1&parent=26&child=1

API error 1 (datastore_v3: BAD_REQUEST): operating on too many entity groups in a single transaction.

Case2 トランザクション内、Entity Groupの制約

結果は

child \ parent	1	2	25	26	30
1	○	○	○	×	×
10	○	○	○	×	×
50	○	○	○	×	×
100	○	○	○	×	×

Case2 トランザクション内、Entity Groupの制約

cross group transactionsはsingle group transactionsとは違うのだよ

```
// TransactionOptions are the options for running a transaction.
type TransactionOptions struct {
    // XG is whether the transaction can cross multiple entity groups. In
    // comparison, a single group transaction is one where all datastore keys
    // used have the same root key. Note that cross group transactions do not
    // have the same behavior as single group transactions. In particular, it
    // is much more likely to see partially applied transactions in different
    // entity groups, in global queries.
    // It is valid to set XG to true even if the transaction is within a
    // single entity group.
    XG bool
    // Attempts controls the number of retries to perform when commits fail
    // due to a conflicting transaction. If omitted, it defaults to 3.
    Attempts int
```

Case3 トランザクションの有効期限

https://cloud.google.com/appengine/docs/standard/go/datastore/transactions

Using transactions

A *transaction* is a set of Cloud Datastore operations on one or more entities. Each transaction is guaranteed to be atomic, which means that transactions are never partially applied. Either all of the operations in the transaction are applied, or none of them are applied. Transactions have a maximum duration of 60 seconds with a 10 second idle expiration time after 30 seconds.

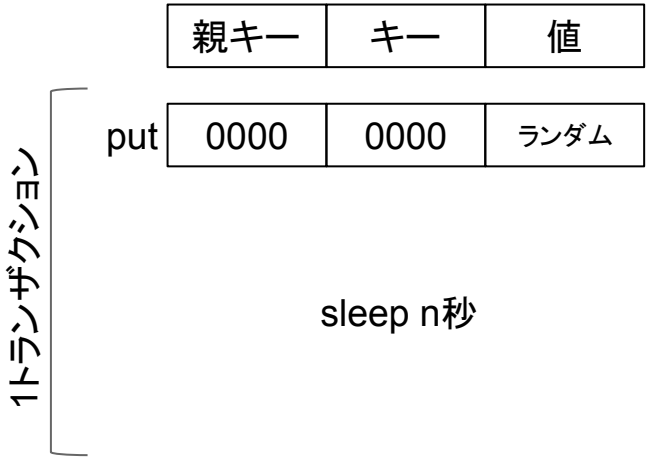
An operation might fail when:

- Too many concurrent modifications are attempted on the same entity group.
- The transaction exceeds a resource limit.
- Cloud Datastore encounters an internal error.

In all these cases, the Cloud Datastore API returns an error.

Case3 トランザクションの有効期限

task queueで・・・



n	結果
1	○
10	○
50	○
70	

n = 70の場合にどうなるか

Case3 トランザクションの有効期限

https://xxx.appspot.com/case3?n=70

Case3 トランザクションの有効期限

結果は

n	結果
1	○
10	○
50	○
70	×

エラーが返却されるのは、RunInTransactionから。上記の場合、
70秒後にエラーが返される

Case4 task queueはroll back対象？

1トランザクション

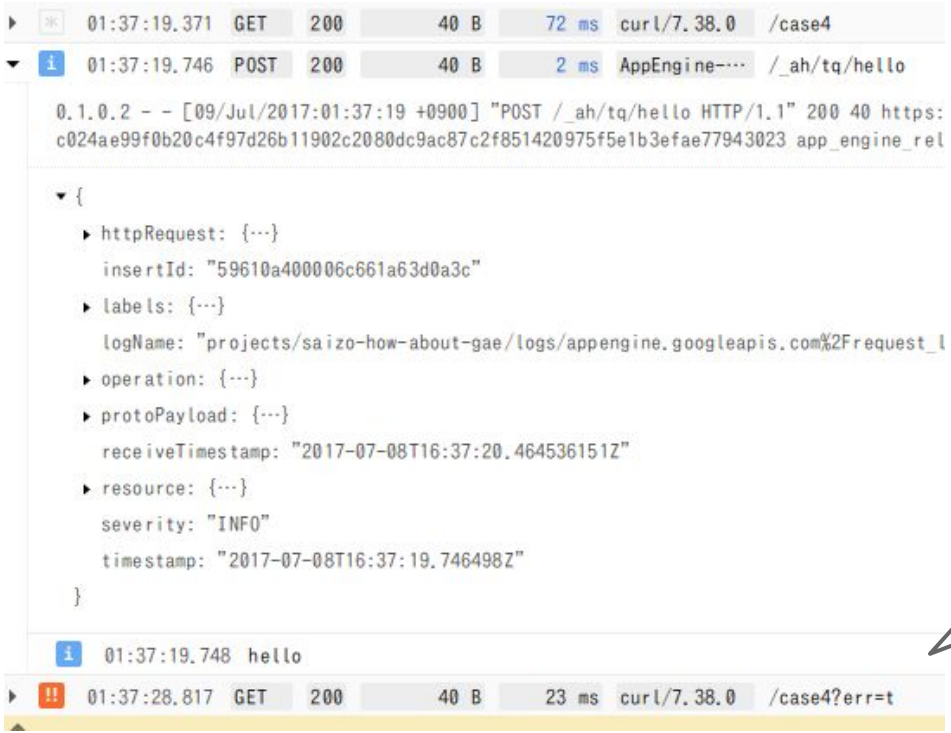
task queueの登録 ⇒ logにhelloを出力

強制的にエラーを返す

err	結果
なし	○
t	

Case4 task queueはroll back対象？

https://xxx.appspot.com/case4?err=t



/_ah/tq/helloは呼ばれない