Distributed System Reading Notes

Week 4

第六組
組員：梁耕銘(110062704)周崑水(108005514)劉新正(109065851)

Table of Contents

1. The Potential Dangers of Causal Consistency and an Explicit Solution

CONTRIBUTERS: 梁耕銘

Introduction (https://hackmd.io/XoiKxs9IT_2Vx3_smJHWqw)

-梁耕銘

Improvement, limitation and weakness (https://hackmd.io/axz1q53VS-2xgTZUgDXLjA?both)

-周崑水

2. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System

CONTRIBUTERS: 劉新正

3. Lineraizability: A Correctness Condition for Concurrent Objects

CONTRIBUTERS: 周崑水

ACM Transactions on Programming Languages and Systems, Vol No. 3. July 1990

CONTRIBUTERS: 周崑水

`Distributed System`

Outline

若有一個資料物件 (Data Object) 被一個以上的同步程序 (concurrent Processes) 共用，則稱為同步物件 (concurrent object)。藉由探索同步物件其抽象資料型別的語意 (Exploit the semantics of abstract data types) (*這裡無法確定作者所說的抽象型別的語意指的是什麼?)得到是否具有線性化 (Lineraizability) 這個可以進行修正的條件(correctness condition) 。
此 paper 作者用形式化的方式定義了的 linearizability, 並且與其他的修正條件 (Correctness condtions)做比較，並展現了一個可以證明導入的正確性的方法。以及說明如何推理出可線性化的物件。

Key Concept

Concurent objects

A concurrent object is a data object shared by concurrent processes

Lineariazble

A concurrent computation is linearizable if it is “equivalent” to a legal sequential computation

Linearizability is a correctness condition

for concurrent objects that exploits the semantics of abstract data types

The counter part is sequential consistency or serializabilty

Whether a concurrent history is acceptable

it is necesary to take into account the object's inteneded semantics:

- FIFO queue would not be acceptable for stacks, sets, directories, etc.

a legal sequential computation

A history H is sequential if:
(1) The first event of H is an invocation
(2) Each invocation, except possibly the last, is immediately followed by a matching response. Each response is immediately followed by a matching invocaion.
A history that is not sequential is concurrent.

Well-formed history

A history H is well-formed if each process subhistory H|P of H is sequential.

Linearizability is a local and nonblocking property

- local property
- nonblocking property
  a pending invocation of a totally-defined opeation is never required to wait for another pending invocation to cmoplete. (一個暫緩的調用只要他是有全面定義的作業...)

Motivation

Two requriements on defining a correctness condition

- First: each operation should appear to "take effect" instantaneously
- Second, the order of nonconcurrent operations should be preserved

Solved Problem & Contribution

- This paper defines linearizability, compares it to other correctness conditions
- Presents and demonstrated a method for providing the correctness of implementation
- Show how to reason about concurrent objects, given they are linerizable

Solutions

Research Value & Application

Institution of the approach

Main Result, evaluated and mean

Improvement, limitation and weakness (Author's future work)

Questions

4. Don't Settle for Eventual: Scalable Causal Consistency for Wide-Area Strage with COPS

CONTRIBUTERS: 梁耕銘

causal+

- a consistency model
- causal consistency with convergent conflict handling
  - replicas never permanently diverge and that conflicting updates to the same key are dealt with identically at all sites
    - clients see only progressively newer versions of keys
  - clients see a **causally-correct, conflict-free, always-progressing data store**

COPS (Clusters of Order-Preserving Servers)

- deliver this consistency model across the wide-area
- key contribution
    - scalability
        - which can enforce causal dependencies between keys stored across an entire cluster
    - central approach
        - tracking and explicitly checking whether causal dependencies between keys are satisfied in the local cluster before exposing writes
- provides causal+ consistency and is designed to support complex online applications
- scalability requirements for ALPS systems
    - previous systems required that all data fit on a single machine or that all data that potentially could be accessed together fit on a single machine
    - data stored in COPS can be spread across an arbitraty-sized datacenter, and dependencies (or get transacctions) can strtch across many servers in the datacenter

overview

- COPS is a key-value storage systeme
    - run across a small number of datacenters
    - each dataenter has a local COPS cluster with a complete replica of the stored data
    -
- each local COPS clusteris set up as a linearizable (strongly consistent) key-value store

SYSTEM COMPONET

- key-value store
    - provides linerazzble operation on keys
    - store in 2 ways
        1. key-value pair has metadata, and metadata is a version number in COPS, in COPS-GT it is both version number and a list of dependency
        2. key-value store export 3 operation
            - get_by_version

        - put_after
        - dep_check
    3. for COPS-GT, keeps around old version of pairs, to ensure providing get transaction
- client library
    - read via get or get_trans
    - write via put

GOALS

- provide causal+ consistency
- minimize overhead of consistency-preserving replication
- (COPS-GT) minimize space requirements
- (COPS-GT) ensure fast get_trans operation

key-value store

> each cluster maintains its own copy of the key-value store

- scalability
    - partitions the keyspace across a cluster's nodes using consistent hashing
- falut tolerance
    - each key is replicated across a small number of nodes using chain replication

COPS-GT

- provides **get transactions** that give clients a consistent view fo multiple keys
    - no locks
    - non-blocking
    - tack at most two parallel rounds of intra-datacenter requests
- furst ALPS system to achieve non-blocking scalable get transactions

ALPS (Availability, low Latency, Partition-tolerance, high Scalability)

- sacrifice strong consistency

causal+ consistency

> causal consistency with convergent conflict handling
> causal+ consistency

- eventual consistency
- restrict consideration
  - a key-value data store
  - 2 basic operations
    - put(key, val)
    - get(key, val)=val
  - equivalent to write and read operations in a shared-memory system

potential causality

- denoted **->**
  1. Execution Thread
     - If a and b are two operations in a single thread of execution, then a -> b if operation a happens before operation b
  2. Gets From
     - If a is a put operation and b is a get operation that returns the value written by a, then a -> b
  3. Transitivity
     - For operations a, b, and c, if a -> b and b -> c, then a -> c
  - does not order concurrent
- handling conflicts
  - last-writes-wins
    - default version of COPS
    - use of more explicit conflict resolution procedures
-

causal+ in COPS

- 2 abstract
  1. versions

    - the different values a key has as the versions of a key
    - denote as $key_{version}$
    - if x_i -> y_j then i < j
    - refer to this as causal+ consistency's progressing property
2. dependencies
    - if x_i -> y_j then x_i must be written before y_j
    - more formally y_j depends on x_i if and only if put(x_i) -> put(y_j)

scalable causality

- Log-exchange-based serialization inhibits replica scalability
    - either causal dependencies between keys are limited to the set of keys that can be stored on one node, or a single node (or replicated state machine) must provide a commit ordering and log for all operations across a cluster

garbage, faults, and conflicts

garbage collection subsystem

- version garbage collection (COPS-GT only)
- dependency garbage collection (COPS-GT only)
- client metadata garbage collection

falut tolerance

- client failures
- key-value node failures
- datacenter failures

confilct detection

conclusion

- high-scale, wide-area provides
    - always on

    - low-latency
- COPS
    - provides causal+ consistency without sacrificing ALPS properties

5. Consistency, Availability, and Convergence

CONTRIBUTERS: 劉新正

6. Consistency and Replication

CONTRIBUTERS: 周崑水

Distributed System Reading Notes

Week 4

第六組
組員：梁耕銘(110062704)周崑水(108005514)劉新正(109065851)

Table of Contents

1. The Potential Dangers of Causal Consistency and an Explicit Solution

CONTRIBUTERS: 梁耕銘

Introduction (https://hackmd.io/XoiKxs9IT_2Vx3_smJHWqw)

-梁耕銘

> trade-off of ds

- causal consistency
  - the strongest achievable model
    - multi-datacenter
  - provides useful semantics for human-facing distributed services

Lurking Dangers, Difficulte Trade-offs

- implementations of caucals consistency face serious scalability challenges
- critical trade-off between write throuput and visibility latency, or the amount of time that each write is hidden from readers due to missing dependencies
  - 2 fators: the number of datacenters and the rate at which each datacenter can check dependencies and apply new writes

Better Living Throug Semantic Context

> practitioners and theoreticians have almost exclusively considered the problem of
> *potential causlaity*
>
> > - each new write causally depends on all writes (versions) that could have influenced it
> > - useful model for closed systems and debugging

- explicit causality
  - a small subset of tranditional potential causality
  - dramtically reducea the depth and degree of the causality graph
  - decrease the number of dependencies per write
    - increase throughput
    - lowers metadata overhead
    - imporves concurrency
  - does not solve problems of all-to-all replication or guaranteed graceful partition tolerance

[Improvement, limitation and weakness](https://hackmd.io/axz1q53VS-2xgTZUgDXLjA?both) (https://hackmd.io/axz1q53VS-2xgTZUgDXLjA?both)

-周崑水

Causal ordering

Causal Ordering (https://www.scattered-thoughts.net/writing/causal-ordering/)

A -> B (event A is causally ordered before evnet B)

- More Forammlly Defined:
  We have a number of machines on which we observe a series of events. These events are either specific to one machine (eg user input) or are communications between machines. We define the causal ordering of these events by three rules:

If A and B happen on the same machine and A happens before B then A -> B
If I send you some message M and you receive it then (send M) -> (recv M)
If A -> B and B -> C then A -> C
We are used to thinking of ordering by time which is a total order - every pair of events can be placed in some order. In contrast, causal ordering is only a partial order - sometimes events happen with no possible causal relationship i.e. not (A -> B or B -> A).

Causual Consistency

如果對一個資料物件的讀寫順序是 causal order, 那麼可以說是因果一致性。

Improvement, limitation and weakness

Improvement:

這篇 Paper 探討要達到 ‘Convergent causal consistency’(收斂因果一致性 or causal+ consistency)的可能問題。加上 convergent 的條件，是因為若只有說 ‘causal consistency’ 是沒有保證存活的，也就是若沒有 convergent, 可以永遠都不要在代理服務器之間傳遞異動。一樣滿足’causal consistency’。

作者提到的 limitations 有三點：

1. 明確因果有其極限 (Explicit causality has limitaion); 作者舉的例子是現實中的人不知道最終何時一致。
2. 維護因果的資料模型是有價的 (Not free for data model)

3. 具有收斂性的明確因果需進行 all-to-all replication (3-2) 會面臨吞吐的瓶頸。
   (3-2) Potntial Danger: Sustainable write throughput is limited to the slowest datacenter, so adding datacenters does not increase throughput, Simutaneously scaling writes with datacenters requires quadratic server capacity, and violating this limit leads to arbitrarily high visibility latency.

Weakness & Questions

作者針對 convergent causal consistency 的其中一個問題：追蹤潛在相依(potential dependency)，提出的解決方法是從具有重要性的潛在相依入手。具體做法是在 application level 將重要的 dependency 記錄下來。並宣稱這樣的做法在一些 Human-facing application 有數量上有明顯減少的優勢。提出的疑問有：

- 作者似乎沒有說明或定義或區分何為 “重要的（matter）” 的 potential dependecy
- Human-facing application 是哪一類系統？
- 只有提到有數量的下降，但沒有看到有實驗數據。
  這邊說的數量下降指的是在 6.conclusion 提到
  “we can decrease per-operation stroage and processing costs via application level explicit causality”

2. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System

CONTRIBUTERS: 劉新正

3. Lineraizability: A Correctness Condition for Concurrent Objects

CONTRIBUTERS: 周崑水

ACM Transactions on Programming Languages and Systems, Vol No. 3. July 1990

CONTRIBUTERS: 周崑水

`Distributed System`

Outline

若有一個資料物件 (Data Object) 被一個以上的同步程序 (concurrent Processes) 共用，則稱為同步物件 (concurrent object)。藉由探索同步物件其抽象資料型別的語意 (Exploit the semantics of abstract data types) (*這裡無法確定作者所說的抽象型別的語意指的是什麼?)得到是否具有線性化 (Lineraizability) 這個可以進行修正的條件(correctness condition) 。
此 paper 作者用形式化的方式定義了的 linearizability, 並且與其他的修正條件 (Correctness condtions)做比較，並展現了一個可以證明導入的正確性的方法。以及說明如何推理出可線性化的物件。

Key Concept

Concurent objects

A concurrent object is a data object shared by concurrent processes

Lineariazble

A concurrent computation is linearizable if it is “equivalent” to a legal sequential computation

Linearizability is a correctness condition

for concurrent objects that exploits the semantics of abstract data types

The counter part is sequential consistency or serializabilty

Whether a concurrent history is acceptable

it is necesary to take into account the object’s inteneded semantics:

- FIFO queue would not be acceptable for stacks, sets, directories, etc.

a legal sequential computation

A history H is sequential if:
(1) The first event of H is an invocation
(2) Each invocation, except possibly the last, is immediately followed by a matching response. Each response is immediately followed by a matching invocaion.
A history that is not sequential is concurrent.

Well-formed history

A history H is well-formed if each process subhistory H|P of H is sequential.

Linearizability is a local and nonblocking property

- local property

- nonblocking property
  a pending invocation of a totally-defined opeation is never required to wait for another pending invocation to cmoplete. (一個暫緩的調用只要他是有全面定義的作業...)

Motivation

Two requriements on defining a correctness condition

- First: each operation should appear to “take effect” instantaneously
- Second, the order of nonconcurrent operations should be preserved

Solved Problem & Contribution

- This paper defines linearizability, compares it to other correctness conditions
- Presents and demonstrated a method for providing the correctness of implementation
- Show how to reason about concurrent objects, given they are linerizable

Solutions

Research Value & Application

Institution of the approach

Main Result, evaluated and mean

Improvement, limitation and weakness (Author’s future work)

Questions

4. Don’t Settle for Eventual: Scalable Causal Consistency for Wide-Area Strage with COPS

CONTRIBUTERS: 梁耕銘

causal+

- a consistency model
- causal consistency with convergent conflict handling
    - replicas never permanently diverge and that conflicting updates to the same key are dealt with identically at all sites
        - clients see only progressively newer versions of keys
    - clients see a **causally-correct, conflict-free, always-progressing data store**

COPS (Clusters of Order-Preserving Servers)

- deliver this consistency model across the wide-area
- key contribution
    - scalability
        - which can enforce causal dependencies between keys stored across an entire cluster
    - central approach
        - tracking and explicitly checking whether causal dependencies between keys are satisfied in the

local cluster before exposing writes

- provides causal+ consistency and is designed to support complex online applications
- scalability requirements for ALPS systems
    - previous systems required that all data fit on a single machine or that all data that potentially could be accessed together fit on a single machine
    - data stored in COPS can be spread across an arbitraty-sized datacenter, and dependencies (or get transacctions) can strtch across many servers in the datacenter

overview

- COPS is a key-value storage systeme
    - run across a small number of datacenters
    - each dataenter has a local COPS cluster with a complete replica of the stored data
    -
- each local COPS clusteris set up as a linearizable (strongly consistent) key-value store

SYSTEM COMPONET

- key-value store
    - provides linerazzble operation on keys
    - store in 2 ways
        1. key-value pair has metadata, and metadata is a version number in COPS, in COPS-GT it is both version number and a list of dependency
        2. key-value store export 3 operation
            - get_by_version
            - put_after
            - dep_check
        3. for COPS-GT, keeps around old version of pairs, to ensure providing get transaction
    - client library
        - read via get or get_trans
        - write via put

GOALS

- provide causal+ consistency
- minimize overhead of consistency-preserving replication
- (COPS-GT) minimize space requirements
- (COPS-GT) ensure fast get_trans operation

key-value store

> each cluster maintains its own copy of the key-value store

- scalability
  - partitions the keyspace across a cluster's nodes using consistent hashing
- falut tolerance
  - each key is replicated across a small number of nodes using chain replication

COPS-GT

- provides **get transactions** that give clients a consistent view fo multiple keys
  - no locks
  - non-blocking
  - tack at most two parallel rounds of intra-datacenter requests
- furst ALPS system to achieve non-blocking scalable get transactions

ALPS (Availability, low Latency, Partition-tolerance, high Scalability)

- sacrifice strong consistency

causal+ consistency

> causal consistency with convergent conflict handling causal+ consistency

- eventual consistency
- restrict consideration

    - a key-value data store
    - 2 basic operations
        - put(key, val)
        - get(key, val)=val
    - equivalent to write and read operations in a shared-memory system

potential causality

- denoted **->**
    1. Execution Thread
        - If a and b are two operations in a single thread of execution, then a -> b if operation a happens before operation b
    2. Gets From
        - If a is a put operation and b is a get operation that returns the value written by a, then a -> b
    3. Transitivity
        - For operations a, b, and c, if a -> b and b -> c, then a -> c
    - does not order concurrent
- handling conflicts
    - last-writes-wins
        - default version of COPS
        - use of more explicit conflict resolution procedures
-

causal+ in COPS

- 2 abstract
    1. versions
        - the different values a key has as the versions of a key
        - denote as \(key_{version}\)
        - if \(x_i\) -> \(y_j\) then i < j
        - refer to this as causal+ consistency’s progressing property
    2. dependencies

        - if \(x_i\) -> \(y_j\) then \(x_i\) must be written before \(y_j\)
        - more formally \(y_j\) depends on \(x_i\) if and only if put(\(x_i\)) -> put(\(y_j\))

scalable causality

- Log-exchange-based serialization inhibits replica scalability
    - either causal dependencies between keys are limited to the set of keys that can be stored on one node, or a single node (or replicated state machine) must provide a commit ordering and log for all operations across a cluster

garbage, faults, and conflicts

garbage collection subsystem

- version garbage collection (COPS-GT only)
- dependency garbage collection (COPS-GT only)
- client metadata garbage collection

falut tolerance

- client failures
- key-value node failures
- datacenter failures

confilct detection

conclusion

- high-scale, wide-area provides
    - always on
    - low-latency
- COPS
    - provides causal+ consistency without sacrificing ALPS properties

5. Consistency, Availability, and Convergence

CONTRIBUTERS: 劉新正

6. Consistency and Replication

CONTRIBUTERS: 周崑水