INFO8002

Large-Scale Data Systems

Exercise Session #3

Academic year 2021-2022

Reminder

REMINDER :

Consensus

- **Consensus** mechanisms allows a distributed system to maintain some common state or to agree on some future actions.

- **FLP Theorem**

*“In an asynchronous network where messages may be delayed but not lost, there is **no consensus algorithm** that **is guaranteed to terminate** in every execution for all starting conditions, if at least one node may experience failure.”*

REMINDER :

Consensus VS Uniform Consensus

- Two types of consensus algorithms, differing on the guarantees they provide in the presence of faulty processes:

Module:
Name: *Consensus*, **instance** *c*.

Events:
Request: < c, Propose | **v** > **:** Propose value **v** for consensus.
Indication: < c, Decide | **v** > **:** Outputs a decided value **v** of consensus.

Properties:
C1: *Termination: "Every correct process eventually decides some value."*
C2: *Validity: "If a process decides* ***v****, then,* ***v*** *was proposed by some process."*
C3: *Integrity: "No process decides twice."*
C4: *Agreement: "No two correct process decide differently."*

Module:
Name: *UniformConsensus*, **instance** *uc*.

Events:
Request: < uc, Propose | **v** > **:** Propose value **v** for consensus.
Indication: < uc, Decide | **v** > **:** Outputs a decided value **v** of consensus.

Properties:
C1: *Termination: "Every correct process eventually decides some value."*
C2: *Validity: "If a process decides* ***v****, then,* ***v*** *was proposed by some process."*
C3: *Integrity: "No process decides twice."*
C4: Uniform *Agreement: "No two processes decide differently."*

REMINDER :

Consensus VS Uniform Consensus

- **Consensus** can be implemented in synchronous and partially synchronous systems:

Consensus:

- **Fail-stop - Hierarchical**
  - Broadcast: **Best effort**
  - Failure Detector : **Perfect**

Uniform Consensus:

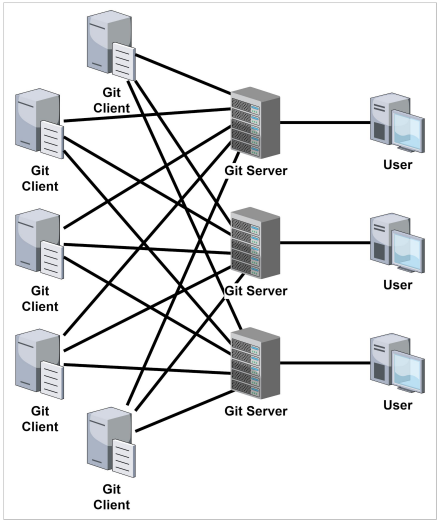
- **Fail-stop - Hierarchical**
  - Broadcast: **Reliable & Best effort**
  - Failure Detector : **Perfect**

- **Fail-noisy - Leader Driven**
  - Use of **Epoch Consensus** using **Eventual Leader Election**.

PROBLEM 1

Version Control System

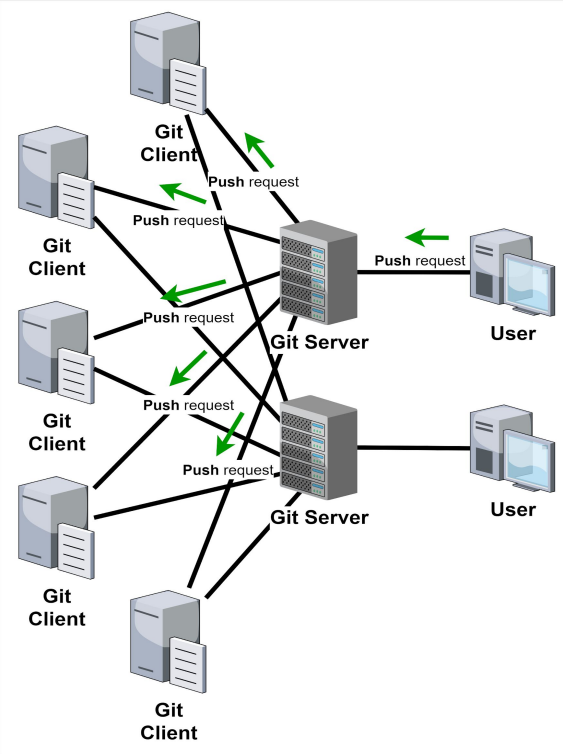
You are responsible for designing a system allowing decentralized version control.



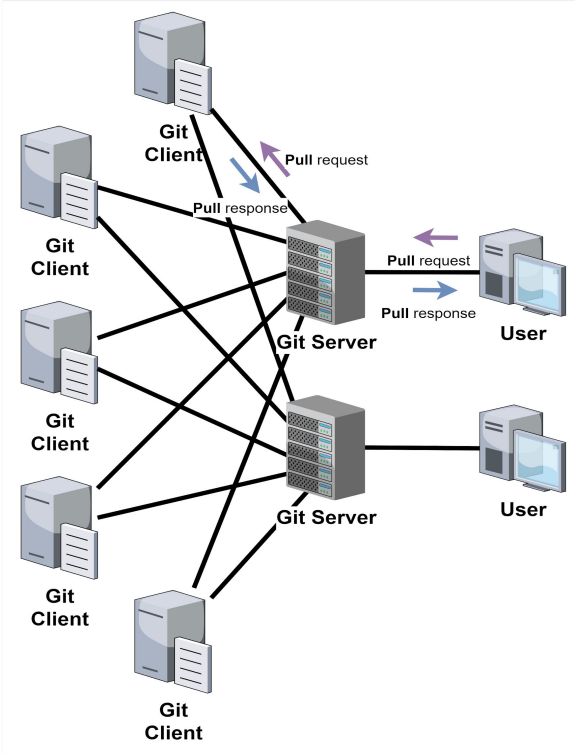
Specify an architecture for this distributed version control system and **provide** a pseudo-implementation using consensus.

Version Control System

Write query



Read query



Version Control System

Problems:

1. **How can we ensure that operations terminates ?**

Version Control System

Problems:

1. **How can we ensure that operations terminates ?**

   Property n°1: **"Termination"**

   "If a correct process invokes an operation, then the operation eventually completes."

2. **How can we ensure that commits are applied consistently on all remote clients?**

Version Control System

Problems:

1. **How can we ensure that operations terminates ?**

   Property n°1: **"Termination"**

   "If a correct process invokes an operation, then the operation eventually completes."

2. **How can we ensure that commits are applied consistently on all remote clients?**

   Property n°2: **"Agreement"**

   "All correct processes applies the same sequence of operation."

Version Control System

Module Specification:

Module 1: Interface and properties of version control system

Module:

Name: *GitClient* **, instance** *gc***.**

Events:

Request: < gc, Push | m, n > **:** Invokes a **push** operation of commit **m** with commit id **n**.

Request: < gc, Pull | n > **:** Invokes a **pull** operation starting at commit id **n.**

Indication: < gc, PushReturn | n > **:** Completes a **push** operation with commit **n**.

Indication: < gc, PullReturn | m > **:** Completes a **pull** operation with commit list **m**.

Properties:

G1: Termination.

G2: Agreement. } *"Total Order Broadcast"* Module

Version Control System

Implementation

Algorithm 1:

Implements:
GitClient, **instance** *gc*.

Uses:
ReliableBroadcast, **instance** *rb*.
Consensus (multiple instances)

upon event < gc, Init >**do**
???

upon event < gc, Pull | n > **do**
???

upon event < gc, Push | m, n > **do**
???

upon event < *rb*, Deliver | m, n> **do**
???

upon event *undecided* ≠ ⌀ **and** *wait* = **False do**
???

upon event < *c.r*, Decide | decided > **do**
???

Version Control System

Implementation

Algorithm 1:

Implements:
GitClient, **instance** *gc*.
Uses:
ReliableBroadcast, **instance** *rb*.
Consensus (multiple instances)

upon event < gc, Init >**do**
commits := *undecided* := ∅;
round := **1**
wait := **False**

upon event < gc, Pull | n > **do**
trigger < gc, PullReturn | commits[**n**:**end**] >

upon event < gc, Push | m, n > **do**
trigger < *rb*, Broadcast | <*m,n*> >;

upon event < *rb*, Deliver | m, n> **do**
if *m*∉ *undecided* **then**
undecided := *undecided* ∪ {*m, n*};

upon event *undecided* ≠ ∅ **and** *wait* = **False do**
wait := **True**
trigger < *c.round*, Propose | undecided >;

upon event < *c.r*, Decide | decided > **do**
forall (m, n) ∈ sort(decided) **do** // Sort by commit n°.
commits := commits ∪ m; // Apply commit n° **n.**
trigger < gc, PullReturn | n >
undecided := undecided \ *decided;*
round := *round* + 1;
wait := **False;**

Content Sharing System

Problems:

- Which consensus algorithm would you use
    1. In a **fail-stop** system?
    2. In a **fail-silent** system?
    3. In a **byzantine** system?

- Is Uniform Consensus necessary? When ?

HOMEWORK !