Distributed Snapshot for Rollback-Recovery with One-Sided Communications

Franck Butelle, *Camille Coti*

LIPN, CNRS UMR 7030, SPC, Université Paris 13, France

HPCS 2018
July 18th, 2018, Orléans, France

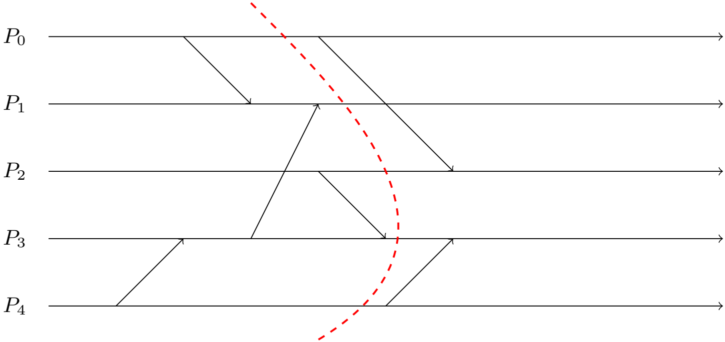
Outline

Distributed snapshot

Goal : store a **consistent state** of the system

- Take a checkpoint of each process
- Get a **consistent cut**
- No message is crossing the cut

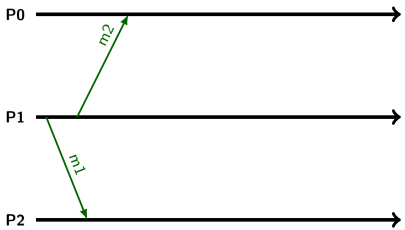
-> Problem : synchronize the processes



Chandy & Lamport's algorithm (1985)

Idea : circulate a marker

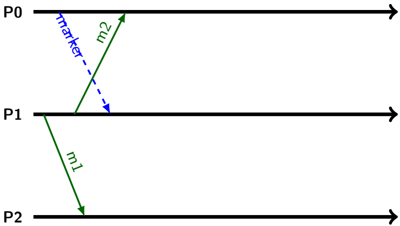
- Initiate the checkpoint wave by **sending a first marker**
- Once a process receives the marker :
  - **Flush** the communication channels
  - Take a **local snapshot**
  - **Send the maker** to all the other processes
- Checkpoint wave done (locally) after reception of all the other processes' markers.



Chandy & Lamport's algorithm (1985)

Idea : circulate a marker

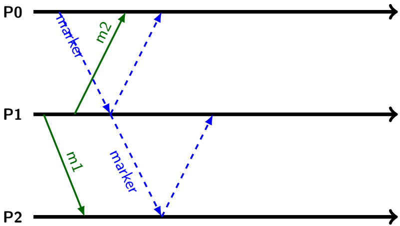
- Initiate the checkpoint wave by **sending a first marker**
- Once a process receives the marker :
  - **Flush** the communication channels
  - Take a **local snapshot**
  - **Send the maker** to all the other processes
- Checkpoint wave done (locally) after reception of all the other processes' markers.



Chandy & Lamport's algorithm (1985)

Idea : circulate a marker

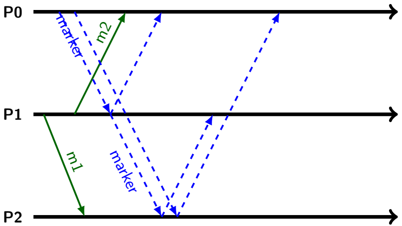
- Initiate the checkpoint wave by **sending a first marker**
- Once a process receives the marker :
  - **Flush** the communication channels
  - Take a **local snapshot**
  - **Send the maker** to all the other processes
- Checkpoint wave done (locally) after reception of all the other processes' markers.



Chandy & Lamport's algorithm (1985)

Idea : circulate a marker

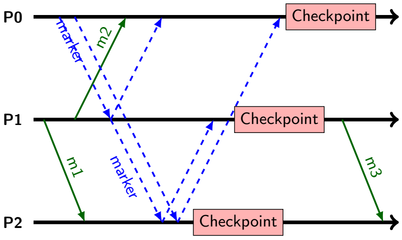
- Initiate the checkpoint wave by **sending a first marker**
- Once a process receives the marker :
  - **Flush** the communication channels
  - Take a **local snapshot**
  - **Send the maker** to all the other processes
- Checkpoint wave done (locally) after reception of all the other processes' markers.



Chandy & Lamport's algorithm (1985)

Idea : circulate a marker

- Initiate the checkpoint wave by **sending a first marker**
- Once a process receives the marker :
  - **Flush** the communication channels
  - Take a **local snapshot**
  - **Send the maker** to all the other processes
- Checkpoint wave done (locally) after reception of all the other processes' markers.



Communications during the checkpoint wave

"Flush" ? ? ?

- What happens with the **communication channels** during the checkpoint wave ?
- Two possible interpretations :
  - **Log** the messages sent during the wave (*Lemarinier et al, Cluster 2004*)
  - **Block** the messages until the end of the wave (*Coti et al, SC 2006*)

Why does it work ?

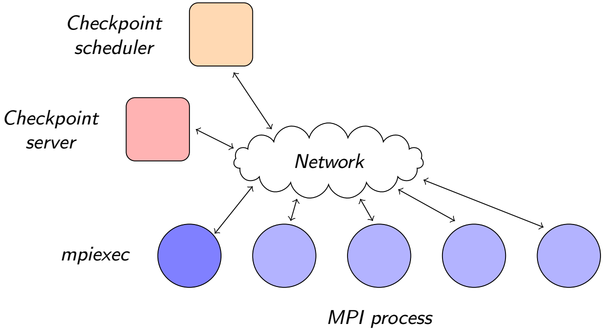
- Communication channels ave the **FIFO property**
- Messages do not pass the markers (and vice versa)

Application to fault-tolerance

Can be used for **fault tolerance**

- Store the checkpoints on a reliable storage support
- **Rollback** on the checkpoints after a process failure

Example : implementation in MPICH-V [1]

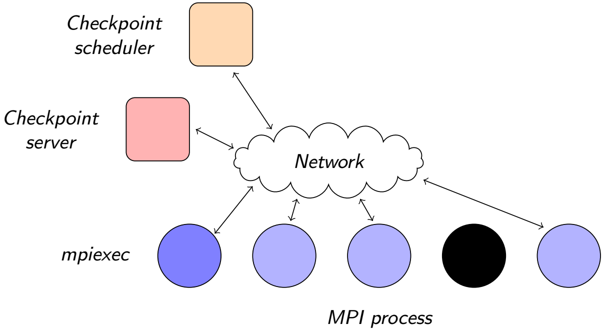


Application to fault-tolerance

Can be used for **fault tolerance**

- Store the checkpoints on a reliable storage support
- **Rollback** on the checkpoints after a process failure

Example : implementation in MPICH-V[1]

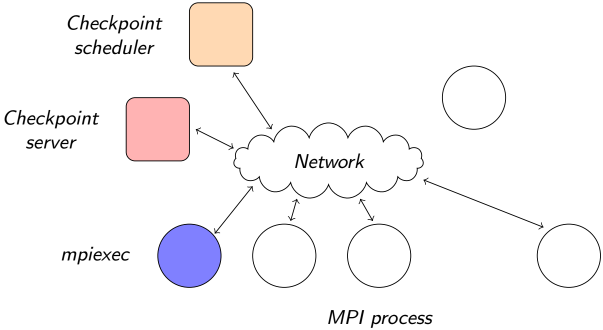


Application to fault-tolerance

Can be used for **fault tolerance**

- Store the checkpoints on a reliable storage support
- **Rollback** on the checkpoints after a process failure

Example : implementation in MPICH-V [1]

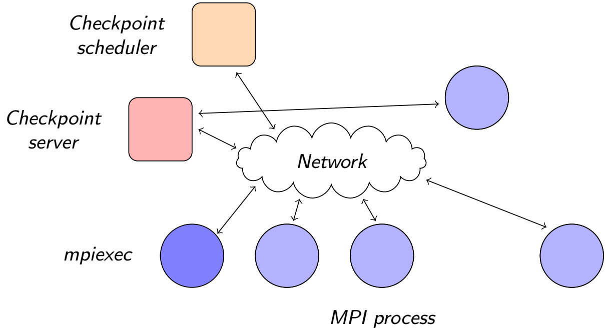


Application to fault-tolerance

Can be used for **fault tolerance**

- Store the checkpoints on a reliable storage support
- **Rollback** on the checkpoints after a process failure

Example : implementation in MPICH-V [1]

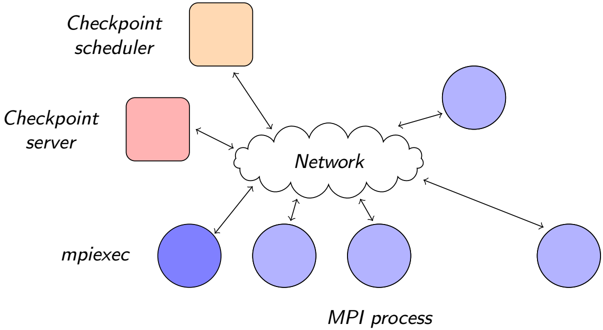


Application to fault-tolerance

Can be used for **fault tolerance**

- Store the checkpoints on a reliable storage support
- **Rollback** on the checkpoints after a process failure

Example : implementation in MPICH-V[1]



1. `http://mpich-v.lri.fr`

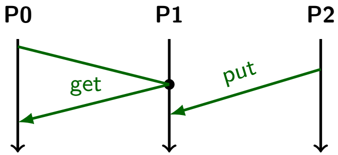
One-sided communication model

Only **one process** takes active part of the communication

- The **source** process
- Other process : **target** process

Two communication primitives

- **put()** : the *source* process **writes** into the *target* process's memory
- **get()** : the *source* process **reads** from the *target* process's memory



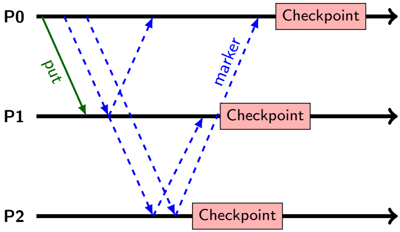
Implementations : RDMA NICs (InfiniBand...), PGAS languages, OpenSHMEM, MPI one-sided communications...

Problem

Now, **distributed snapshot with one-sided communications ?**

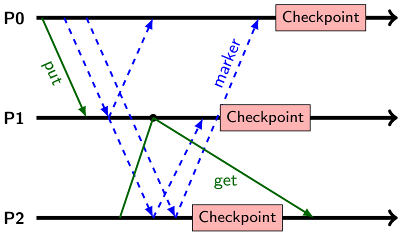
Problem

Now, **distributed snapshot with one-sided communications ?**



Problem

Now, **distributed snapshot with one-sided communications ?**



→ The return of the get() crosses the checkpoint line

- The cut is **not consistent**

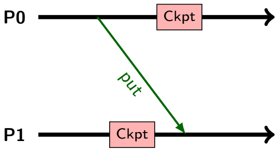
What do we want to avoid ?

Messages **crossing** the wave

- A message sent **before** the source takes its checkpoint is received **after** the target has taken its checkpoint.

Why is it a problem ?

- Breaks consistency



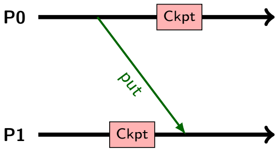
What do we want to avoid ?

Messages **crossing** the wave

- A message sent **before** the source takes its checkpoint is received **after** the target has taken its checkpoint.

Why is it a problem ?

- Breaks consistency

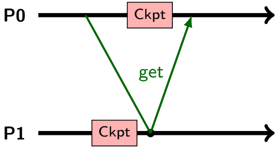


Messages **overlapping** the wave

- A message request sent **before** the source takes its checkpoint is completed **after** the checkpoint
- ... but the source is reached **after** it has taken its own checkpoint.

Is it a problem ?

- Depends on what is stored in the checkpoint



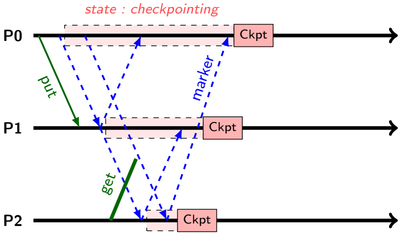
Message delay

Switch into **checkpointing state** upon reception of the first marker

- Switch back to normal state **after** completion of the checkpoint wave.
- **Delay** communication requests while in checkpointing state.

How can it be implemented ?

- *e.g.*, on Verbs/InfiniBand : modify the queue pair's receiving state.



Properties :

- Can overlap the checkpoint wave ✗ or ✓
- Cannot cross it ✓

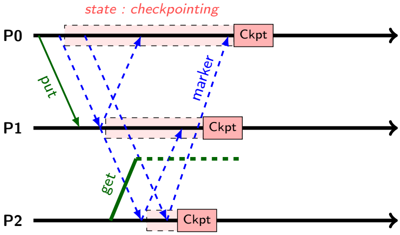
Message delay

Switch into **checkpointing state** upon reception of the first marker

- Switch back to normal state **after** completion of the checkpoint wave.
- **Delay** communication requests while in checkpointing state.

How can it be implemented ?

- *e.g.*, on Verbs/InfiniBand : modify the queue pair's receiving state.



Properties :

- Can overlap the checkpoint wave ✗ or ✓
- Cannot cross it ✓

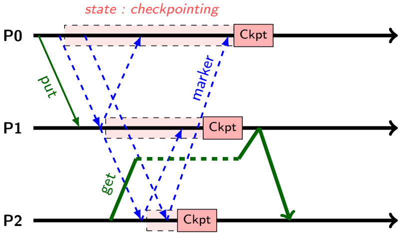
Message delay

Switch into **checkpointing state** upon reception of the first marker

- Switch back to normal state **after** completion of the checkpoint wave.
- **Delay** communication requests while in checkpointing state.

How can it be implemented ?

- *e.g.*, on Verbs/InfiniBand : modify the queue pair's receiving state.



Properties :

- Can overlap the checkpoint wave ✗ or ✓
- Cannot cross it ✓

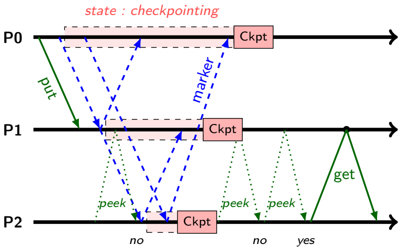
Peek-and-get

Switch into **checkpointing state** upon reception of the first marker

- Switch back to normal state **after** completion of the checkpoint wave.
- Before a *get()* communication : **peek** to see if the target is ready.

If the target switches into checkpointing state **between peek and *get()*** :

- The *get()* returns an error.



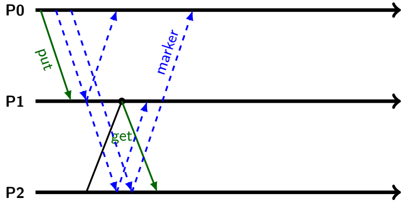
Properties :

- Cannot overlap the checkpoint wave ✔
- Cannot cross it ✔

Double barrier

Perform **two barriers**

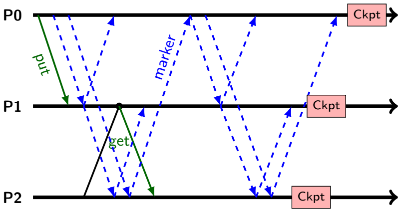
- **First one** : circulation of the marker
- Can be crossed by a *get()*
- Therefore, **second one**
- Stop communicating upon reception of the first maker
- Checkpoint **after completion of the second barrier**



Double barrier

Perform **two barriers**

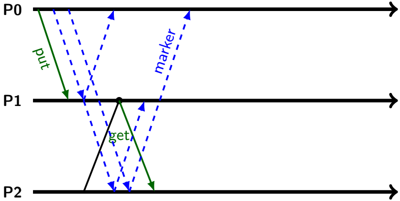
- **First one** : circulation of the marker
- Can be crossed by a *get()*
- Therefore, **second one**
- Stop communicating upon reception of the first maker
- Checkpoint **after completion of the second barrier**



Double barrier (optimized)

Perform this extra synchronizing communication on **processes with a pending *get()*** only

- Fewer messages
- Sufficient to ensure **communication channel flushing**



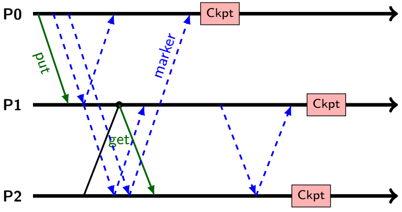
Properties :

- Cannot overlap the checkpoint wave ✔
- Cannot cross it ✔

Double barrier (optimized)

Perform this extra synchronizing communication on **processes with a pending *get()*** only

- Fewer messages
- Sufficient to ensure **communication channel flushing**



Properties :

- Cannot overlap the checkpoint wave ✔
- Cannot cross it ✔

Comparison

Implementation level

- Double barrier : in the **checkpointing protocol**
- Peek-and-get and delay : either in the protocol or in the driver
  - Peek-and-get : in the **communication routine**
  - Delay : in the **state of the queue pair**

Number of messages

- Double barrier : $n(n-1)$ additional messages (x2)
  - Optimized : 2 additional messages per pending *get()*
- Peek-and-get : many additional messages, until the end of the checkpoint wave
- Delay : no additional messages, requires intervention on the driver

Properties

	Overlap	Cross
Vanilla	✗	✗
Delay	✗	✓
Peek-and-get	✓	✓
Double barrier	✓	✓

Conclusion

Distributed snapshot

- Used to get a **global state** of a distributed system
- Requires specific care with communication channels
- **Chandy & Lamport's algorithm** : checkpoint wave, process synchronization

One-sided communications

- Only the source process takes an active part of the communication
- Primitives : **put()** and **get()**
- *put()* in one message
- *get()* in two messages : request and data
- RDMA : MPI3, OpenSHMEM, UPC...

Chandy & Lamport's algorithm checkpoint wave **crossed by *get()***

- Three algorithm for synchronization during the checkpoint wave
- **Delay, peek-and-get, double barrier**
- Different levels of implementation
- Different overhead

-> Next : implementation and **performance evaluation**