On distributed recovery for SPMD deterministic HPC applications

Tatiana V. Martsinkevich, Thomas Ropars, Amina Guermouche, Franck Cappello
• Number of cores on one CPU and number of CPU grows
• Can expect frequent hardware failures

• Using a fault tolerance protocol is a must

• Many protocols already exist
• Hybrid protocols are the most promising
Motive for this talk: HydEE

- HydEE – a hybrid hierarchical rollback-recovery protocol for message passing applications

- Divide processes in groups (clusters)
  - Coordinated checkpointing within the cluster
  - Message logging between clusters
    - Sender-side logging

- Assumption: send-deterministic applications
• In any correct execution:
  – Same messages are always sent in the same order
  – The reception order has no impact on the execution
1. All processes inside C2 rollback to the last checkpoint
2. Others resend logged messages to processes in C2
HydEE: Recovery issues

- Causal dependency between messages

Diagram:

- C1: P0
- C2: P1 → m1 → P2
- C3: P3
- m4 → m5

m5 can be received by mistake before m2
HydEE: Recovery issues

- Causal dependency between messages

  - Use phases to express dependency
    - Update my phase when intra-cluster message received
    - Update and increment when message comes from another cluster

- Guaranty of replay of orphan messages
  - Send-determinism guarantees that the same message will be replayed by the rolled back process
HydEE: Recovery process

- A separate recovery process to orchestrate the recovery
- It ensures causal order: no message is sent until there are orphan messages in lower phase
- It has the info about
  - The phase to which process rolls back
  - Phases of all logged messages to be replayed
  - Number of orphan messages in each phase

Diagram:

- C1
- C2
- C3
- P0
- P1
- P2
- P3
- m1
- m2
- m3
- m4
- RP

1. m1, m2 replayed
2. send next msg
3. send next msg

Orphan message
• Recovery process can slow down the recovery
  • Process has to wait for the permit from RP to resend the next logged message
• The faster the network the more is impact of the centralized recovery

Actually:
• Restarted process can immediately access logged messages
• It can figure out what messages not to replay
• If it could figure out causal order by itself recovery would finish faster
Assumption about determinism

- Relax the constraints of send-determinism
- One communication consists of: sender, receiver, message content

SPMD-determinism - in any correct execution the set of communications is the same

- Typical property of SPMD applications
• Restarted process gets all the logs and info about orphan messages

• It decides autonomously whether
  – to receive next message from the log
    • which message it should be then?
  – to receive next message from another restarted process
  – the next message to send is an orphan message so no need to resend

• Phases don’t work anymore

Need a mechanism to help the process make the decision
• Main source of confusion: message reception
• Assume that channels are FIFO
  – won’t confuse messages in case of named reception
• Anonymous receptions (MPI_ANY_SOURCE) create problems
Localizing the problem(2)

After rollback P1 receives logs with:

- \(m_3, m_3'\) // from P0
- \(m_4, m_4'\) // from P2

*from the point of view of P1*

```c
for( int ii = 0; ii < num_iter; ii++ ) {
    for( int i = 0; i < nproc; i++ ) {
        if( i != myrank )
            mpi_send( buf1, count, MPI_INTEGER,
                        i, tag0, MPI_COMM_WORLD );
    }
    for( int i = 0; i < nproc - 1; i++ ) {
        mpi_recv( buf2[i], count, MPI_INTEGER,
                    MPI_ANY_SOURCE, tag0,
                    MPI_COMM_WORLD, &rreq );
    }
    mpi_barrier( MPI_COMM_WORLD );
}
```
Possible approaches

Goal: express causal dependency between anonymous receptions in one process

- Two approaches:
  1. Count my anonymous receptions and propagate to all processes
  2. Define communication *sections* that would separate anonymous receptions
     a) Adding directives `#SECTION_START` and `#SECTION_END`
        - want to avoid this
     b) Automatic runtime detection of sections
Counting number of anonymous receptions

• Count my own anonymous receptions
• Keep a vector of counters of all the other processes
• Append own copy of vector to each sent message
• Update own copy with each message reception

After rollback:
• Choose msg with the corresponding counter \( \leq \) my current counter
• Works but not scalable 😞
Communication sections (1)

- Section confines matching (by tag) send and recv
- Counter for sections
  - increment upon crossing the border between two sections
  - append to each sent message
- Counter of sent message should match my current counter
- Different counters for different messages tags
for( int ii = 0; ii < num_iter; ii++ ) {
    for( int i = 0; i < nproc; i++) {
        if( i != myrank )
            mpi_send( buf1, count, MPI_INTEGER, i, tag0, MPI_COMM_WORLD );
    }
    for( int i = 0; i < nproc - 1; i++) {
        mpi_recv( buf2[i], count, MPI_INTEGER, MPI_ANY_SOURCE, tag0, MPI_COMM_WORLD, &rreq);
    }mpi_barrier( MPI_COMM_WORLD );
}
Distributed recovery with sections

• After rollback $P1 \rightarrow \text{others}$: “I restart from (tag0, cnt=0)”

• $\text{Others} \rightarrow P1$: “Here is my message log starting from cnt=0:”
  
  $m3(\text{tag0, cnt}=0), m3'(\text{tag0, cnt}=1)$  // from $P0$
  $m4(\text{tag0, cnt}=0), m4'(\text{tag0, cnt}=1)$  // from $P2$

• $\text{Others} \rightarrow P1$: “This I received from you since cnt=0:”
  
  $(\text{tag0, cnt}=0) \rightarrow m1$, $(\text{tag0, cnt}=1) \rightarrow m1'$  // from $P0$
  $(\text{tag0, cnt}=0) \rightarrow m2$, $(\text{tag0, cnt}=1) \rightarrow m2'$  // from $P2$

• In the anonymous reception choose messages with matching counter
• Define calls that can start and end a section
  – and guarantee that matching send and receive are within the same section

<table>
<thead>
<tr>
<th>Can open a section:</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_send</td>
</tr>
<tr>
<td>mpi_isend</td>
</tr>
<tr>
<td>mpi_irecv</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Can close a section:</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_recv</td>
</tr>
<tr>
<td>mpi_wait(rreq)</td>
</tr>
<tr>
<td>mpi_waitall(rreqs)</td>
</tr>
<tr>
<td>mpi_waitany(rreqs)</td>
</tr>
</tbody>
</table>

• In a series of consecutive calls that can open/close the section only the first call will trigger the action

```c
for (int i = 0; i < nproc; i++) {
    mpi_send( buf1, count, MPI_INTEGER, i, tag0, MPI_COMM_WORLD );
}
```

only the first mpi_send will open the section for tag0
Automatic detection of sections (2)

- List of counters for each message tag (associated section)
  - struct { int tag; int cnt; bool isOpened};
- Counter incremented when section is re-opened

```c
for( int ii = 0; ii < num_iter; ii++ ) {  // ii = 0, list of counters empty
  for( int i = 0; i < nproc; i++ ) {
    if( i != myrank )
      mpi_send( buf1, count, MPI_INTEGER, i,  // init cnt and open the section ( tag0, 0, true)
                tag0, MPI_COMM_WORLD );  // attach cnt=0 to the msg
  }
  for( int i = 0; i < nproc - 1; i++ ) {
    mpi_recv( buf2[i], count, MPI_INTEGER,  // first recv closes the section (tag0, 0, false)
               MPI_ANY_SOURCE, tag0,
               MPI_COMM_WORLD, &rreq );
  }
  mpi_barrier( MPI_COMM_WORLD );
}
```

Next loop by ii: increment counter upon reaching first mpi_send.
Automatic detection of sections: Asymmetric case(1)

- Sections are easy to detect if all the processes do the same (SPMD parallelism)
- If the execution is not symmetric the definition of sections collapses

```c
for( int ii = 0; ii < num_iter; ii++ ) {
  if ( myrank < nproc / 2 ) {
    for( int i = nproc / 2 ; i < nproc; i++) {
      mpi_send( buf1, count, MPI_INTEGER, i, 
                  tag0, MPI_COMM_WORLD );
    }
  } else {
    for( int i = 0; i < nproc / 2; i++) {
      mpi_recv( buf2[i], count, MPI_INTEGER, 
                 MPI_ANY_SOURCE, tag0, 
                 MPI_COMM_WORLD, &req[i] );
    }
  }
}
mpi_barrier( MPI_COMM_WORLD );
```

**proc group1:**
mpi_send will open a section but no matching mpi_recv to close it

**proc group 2:**
mpi_recv can only close a section, no matching mpi_send to open it
Automatic detection of sections: Asymmetric case(2)

- Use synchronization calls to detect end of section?
  - it’s possible to write asymmetric program without explicit synchronization (e.g. ping-pong with two tags)

- Re-define set of calls to open and close a section?
  - Two sets overlap →don’t know to which set a call belongs

<table>
<thead>
<tr>
<th>SYMMETRIC</th>
<th>ASYMMETRIC</th>
<th>No solution yet</th>
</tr>
</thead>
</table>
| **Can start a section:** mpi_send
  mpi_isend
  mpi_Irecv | **Can start & end a section:** mpi_send
  mpi_isend
  mpi_Irecv |
| **Can end a section:**   mpi_recv
  mpi_wait(rreq)
  mpi_waitall(rreqs)
  mpi_waitany(rreqs)   | |


• Find a solution for automatic section detection for asymmetric case
• Come up with a completely different approach?