Overhead of using "Spare" Nodes

A. Hori and K. Yoshinaga, et al.

Contents

1. Summary

In this paper, Two aspects, relating to using spare nodes in a future HPC data center, where the **node failures** are expected to be more **frequent** are studied and the authors propose general models for a RESOURCE-MANAGER which is responsible for:

1. Spare node Allocation
   - 2D(1,1), 2D(1,2) , 3D(1,1) … qD(r,s)
2. Spare node Substitution
   - 0D, 1D, 2D … qD sliding

Benchmarks

- **Failure Model**
  - Only node-failures, no network and process failures
- **Evaluation Metric**:
  - Communication performance degradation (#of message-collisions)

- **Network Topologies**:
  - Cartesian, 2D/3D
- **Codes, Communication Pattern**:
  - 5P-stencil
  - 7P-stencil

2. Communication Libraries

Problem statement:

- The MPI is the most used communication library, but upon process failure a fatal error handler is raised, preventing any user-level fault handling to be implemented.

State of the art:

- Two-Existing fault tolerant frameworks

These provide API's so that the modifications to the existing MPI specifications are minimal.

1. ULFM, User Level Fault Mitigation
   - Falanx
   - LFLR, Local Failure Local Recovery
2. PGAS, Partitioned Global address space
   - GVR, Global View Resilience

Issue

1. The authors claim that the above mentioned frameworks have the following shortcomings:
   - Not straightforward to use
   - Using these frameworks, there will be situations where code for handling the 1st-node failure is different from 2nd-node failure
   - Users of parallel execution environment find these complex
2. These must be as-simple as possible, and the complexity must be abstracted within system software.

Proposal:

1. Develop a framework that resembles like ULFM
2. This framework will replace faulty nodes with spare nodes, in such a way so that users do not need to be concerned with how failures are handled

3. This will be an alternative to user-level substitution, aka system-level substitution
4. Come up with a best way to reserve spare nodes and to use them to replace failed nodes efficiently.

`Yet to address` :,

1. The overhead of having spare nodes on hot/hotter `standby`.
2. `Job mitigation` vs `Fault mitigation`, proving which is better.

3. Allocation Algorithm

The question the authors are try to answer here is:
How many nodes are to be maintained as spare for a network with `N-nodes?`

$$R_{qD(r,s)} = 1 - \frac{(A)^r (B)^{q-r}}{N} \tag{1}$$

- $A : N^{1/q} - s$
- $B : N^{1/q}$
- N : number of nodes in the network
- qD : the dimension of the network
- r : number of sides considered as spare sides, $r \leq q$
- s : thickness of the spare sides

Example Node allocations

- `2D(2,2) : 2D network, 2 out of 4 boundaries are spare sides, 2 side thick`
- `2D(4,2) : 2D network, 4 out of 4 boundaries are spare sides, 2 side thick`
- …
- `qD(r,s) : qD network, r out of q boundaries are spare sides, s side thick`

3.1. Claims

1. There are no significant difference 2D(4,1) and 2D(2,2) $\implies$ The geometric locations of these spare nodes are not significant, but the number of spare is the key player
2. The thickness, s does not affect the nature of spare node substitution $\implies$ study only single node thick spare sides.

4. Substitution Algorithm

The question the authors are trying to answer here is:
How to substitute ***`faulty-nodes`*** *with* ***`spare-nodes?`***

For model they come up with, they state the relationship between $C_{max} \iff F_n$

- C_{max} : `Maximum number of possible message collisions`

- F_n : `The number of node failures`

Proposed methods:

- `0D sliding` aka. (`no sliding`)
  - The `faulty node` is simply replaced by a `spare node`
  - cons:
    * large hop distance from the failed node to the spare node
    * high probability of message collisions
    * high communication latency
  - `STATEMENT`: Choose spare with least `Manhattan distance`

```
if(node.fail){
  failed.node <- spare.node((min (manhattan_dist (fail.node, spare.node) ) )
}
```

  - $C_{max} \Longleftrightarrow F_n$ derived

- `1D sliding`
  - In `0D sliding`, even if the closest spare is chosen, the distance is unlikely to be small and `1D method` avoids this problem.
  - `STATEMENT :` When a node fails, instead of `replacing it` with a `spare node`, the nodes of the column or row that include the failed node shift $\rightarrow$ toward a `spare node`
  - Hop count increases by 1, much smaller than `0D-sliding` for 5P communication pattern
  - Superior to `0D-sliding` but has limitations, for eg: cannot handle more than `3 failures` in `2D(2,1)` allocation pattern.
  - $C_{max} \Longleftrightarrow F_n$ derived

- `2D sliding`, `3D sliding` … `qD sliding`
  - `STATEMENT : The rows and columns of the node space are shifted by one unit to empty the row or column of the failed node`
  - F_n

```
if (topology == cartesian)
{
   highest_degree(sliding method) = n(dimensions)
}
```

  - C_{max}
    * messages pass orthogonally through the vacant rows and columns
    * **all** message routes are the same as they were before failure

    * Hop count increased by 1, Message congestion can be avoided.
    * This behavior is independent of the communication pattern of the application

- `Hybrid sliding`
  - The `qD` substitution methods are independent of each other and can be combined
  - The `combination/hybrid` allows the job to to `survive` with a `greater` number of failures
  - `STATEMENT : The order of sliding methods to be applied in a hybrid method should be in d`
    * Notation; `hybrid(2D+1D+0D) = hybrid(all)`
    * Notation: `hybrid(-2D) = hybrid(all - but 2D)`, combining all possible methods except `2D`
    * Order : `2D-sliding` first whenever its possible, Followed by `1D-sliding` and then `0D-sliding`

- Comparison of `sliding-methods`
  - `SNS: Spare Node Substitution` and `SNA: Spare Node Allocation`
  - `SNS=0D & SNA=2D(1,1)` → `message_collisions = f(network.topology) if n(faults) > 6`
  - `SNS=0D & SNA=2D(2,1)` → `5 faults simulated, can Handle as many as spare nodes`
  - `SNS=1D` → `Handles upto 3 faults, faults>3 cannot be handled for all locations`
  - `SNS=2D & SNA=2D(2,1)` → `Handles upto 2 faults`
  - `hybrid(all)` → used through out the paper and compared against `0D-sliding`
  - `hybrid(-x)` → compared against `hybrid(all)`

5. Evaluations

5.1. In Local Lab Simulations

- The authors `test` the above `node-allocation` and `node-substitution` methods in a **lab setting** on their own **`Node Failure Inducing/ Fault Injection Framework`** developed by them for this `study`
- In this `framework`,
  - Every possible combinations of node failures is `simulated`
  - The number of of `message-collisions` in a `5P-communication` are counted at every link.
  - `Highest` number of `message-collision` is reported

- The authors study a sub-set of their proposals on this simulation environment
  In particular they study, the differences between:

  - hybrid(all) and 0D sliding for each node-allocation policy ∈ {2D(2,1) ,3D(2,1) ,3D(2,1)}
  - hybrid(all) and hybrid(-2D) for each node-allocation policy ∈ {2D(2,1) ,3D(2,1) ,3D(2,1)}

- RESULTS;
  - Its not possible to make general statements on which sliding method is better, given an environment setting.
  - The hybrid(all) out performed 0D-sliding for networks of the order 100x100 and 12x12x12 and vice versa for networks of the order 24x24x24, and these sliding methods had variations within these depending on the number of failures simulated on these networks.
  - The authors also keep a track of which of the individual sliding methods ∈ {0D, 1D, 2D, 3D} get deployed when the hybrid(all) sliding method is used, and report this as selection-rate plot.
  - 25% → 0D, 70% → 1D , 5% → 2D.

5.2. In Real HPC settings

- HPC MACHINES
  - 4th of Top500: K-COMPUTER, (6D Cartesian)
  - 11th of Top500: BLUE GEANS/Q (5D Cartesian)
  - 25th of Top500: TSUBAME 2.5 (Fat Tree)

- The experiments done in simulation are redone in the above HPC settings and the results are reported
  - Depending on the machine topology the simulation results vary with the actual HPC results
  - hyb(-2D) and hyb(all) are compared with each other for several(randomly chosen) number of faults, once again depending on the HPC setup chosen one node substitution method out performs the other.

6. Reviews

6.1. Objective

- This article basically proposes a 2-part algorithm for a RESOURCE MANAGER which is responsible for handling failed-nodes.
  1. preallocating nodes as to be part of the spare-node group
  2. replacing failed-nodes with preallocated spare-nodes in a uni-job environment when there are faults.

6.2. Comments

- Page 7, Line 30:
  - The authors say that the number of combinations of failed node is the factorial of the number of failed nodes aka. $NC(F) = F!$, this subsumes the ordering of node-failures, but the experimental setup (simulation or evaluation) do not appear to be taking this into consideration.
- Page 3, Line 12:
  - The authors do not, benchmark the proposals made for handling the node failure against the existing fault tolerant framework.
  - The functional demerits of `GVR`, `ULFM` were not mentioned other than not being user friendly
  - Have the authors considered `Reinit` that appeared in `IJHPCA` last year that makes Link for: New ULFM API's user friendly.
- Page 7, Line 37:
  - The authors need to specify the `basis` of the choices they they make for the simulation setup.
  - For example they choose 100 x 100, 12 x 12 x 12 and 24 x 24 x 24 are there any implicit basis for these choices like simulation time.
- Page 4, Line 52:
  - The definition of 2D(2,1) is clear and intuitive, but its not so for `3D(2,1)`
  - How does the `SNA` differ for `3D(2,1)` and `2D(2,1)`
  - What is the definition for `3D(2,1)` spare node allocation?
- Page 9, Line 13:
  - Having a explicit/concrete expression for `slow down` ratio would benefit the reader
- Page 11, Line 55:
  - What does it mean to have a `3D-sliding` for a `SNA == 2D(2,1)`
- Page 12, Line 9
  - In the discussion section (quite late), the authors mention that a figure 3, that appears very early in the paper the spare node % are for individual jobs and not for multi job environment, this would not have been misleading if mentioned early

- If component failures are common events are we **still** relying on redundancy `(spare-nodes)` to tolerate hardware faults, clear description of the migration sequence would be helpful.
- The paper has considerable number of `Typos` and `repeated` words, there were more than one instance where the `figures` and the `corresponding texts` describing them were not matching.
  - Some of the markings in the Legends in the Figures were not clear, the use of color in the legend would be beneficial for the reader
  - English and writing style needs review as The article would benefit from a close editing.
  - We found it difficult to follow a few of the author's argument due to the many stylistic and grammatical errors and grammatical errors.
  - The $s < N^q$ is a typo, this should be $s < N^q$
  - The authors quote "white boxes" to be present in the figure 6, which is not explicit.