Fault Tolerant Techniques in Mobile Grid Computing: A Survey

Amit Savyanavar¹, Pranav Ghate²

¹,²Department of Computer and engineering, MITCOE, Kothrud, Pune University, Pune, India

Abstract: Computational grid systems are the systems, where a task is distributed in various geographically distributed nodes. An important property of the grid system is Fault tolerance; it is a property of the system which enables a system to operate properly in case of failures. Checkpointing and replication are the two commonly used techniques for fault tolerance. Furthermore, these techniques mitigate the lost work, they commonly migrates a snapshot of the latest correct state of the failed node or share information between the resources to ensure consistency between them. However, these techniques are not full proof as they may introduce some kind of runtime overheads. Increasing fault tolerance means making your system more reliable and available to the users. It is also essential to satisfy the QoS requirement of the computational grids. Reliability and availability are the two greatest challenges in grid due to the unreliable nature of its resources.

Keywords: Fault tolerance, Computational Grids, Checkpointing, mobile grid systems, QoS.

1. Introduction

A system that coordinates resources which are not bound to a centralized control using some general purpose protocols and some significant Quality of Service (QoS) is called a grid. They consist of nodes that are geographically distributed to execute a specific task. The probability of failure increases with increase of grid components or load. This may give rise to process failure, node failures, machine crashes, link failure etc. Together giving rise to failure to complete the job in given deadline, denial of services violating service level agreements. Consequently, expected QoS degrades. [1]

Grid systems are widely used in the areas like academics, industry etc. A new kind of grids called Mobile Grids is receiving growing heed and is becoming an important part in computational grids. It consists of mobile nodes which provide computing resources and facilitate user access to the grid [2]. Similarly mobile grids can aid in utilization of any unutilized resources on the device. On the contrary they provide a chance for mobile nodes to save their own resources and use the resources of the grid.

While executing a task consider a failure occurs, that is a component fails, then previous work will be lost and the application has to be restarted and continue from scratch resulting in time and resource wastage. We need a technique which can prevent or detect these faults. The property of the grid which enables the system to continue operating smoothly while the failure has occurred is termed as Fault Tolerance. A complementary but a different approach to increase the dependability of the system is Fault prevention. This property includes techniques like inspecting the nodes or the resources and eliminates the cause by which a failure or a fault may occur.

When do we say a failure has occurred? It is the time where the system behaves differently than the specified behaviour. The seed of occurrence of a failure is called an error, and a fault is a root cause of a failure. Nevertheless a fault may not certainly generate an error, regardless the same fault may evolve in multiple failures. Likewise a single error may cause multiple failures.

The paper is organised as follows, Section 2 briefly describes some of the recent fault tolerance technique. Section 3 describes some of the findings and Section 4 presents the conclusion.

2. Fault Tolerance Methodologies

As we know fault tolerance is an important property of mobile grid systems, as faults which may occur through dynamic links, less reliable links etc may cause fatal node failure interrupt failure of the whole system. However some techniques or methods are used to overcome these faults. Some of them are listed below.

2.1 Reliability Arrangements

Paul J darbyIII and nian-Feng Tzeng have proposed a reliability middleware driven checkpointing arrangement [1] where the nodes act as providers and consumers. An arrangement is stable when a consumer or provider does not prefer any other provider or consumer, respectively, to its current partner in the relationship. The relationship is defined as:

\[ \text{MH}_i \rightarrow \text{MH}_j \]

Where, \( \text{MH}_i \) is a mobile host working as a provider and \( \text{MH}_j \) is a mobile host working as a dedicated Consumer. Above relationship is defined as, each provider in the arrangement has a dedicated consumer. If stable arrangement which is formed say, \( [\] \), holds the relationship \( R_{i} \geq R_{a} \) where \( R_{i} \) is the reliability of some checkpointing arrangement and \( R_{a} \) is the reliability of the stable checkpointing arrangement. This arrangement is expressed as \( [\] )

\[ \prod_{i \in [\]} [1 - (1 - \omega \epsilon ) (1 - p \epsilon s \epsilon d)] \]

Where, \( [\] \) is some unique checkpointing arrangement. Which is a directed graph consisting of a unique collection \( \text{MH}_i \rightarrow \text{MH}_j \) relationship on \( \text{MH}_i \), where
k ≠ l.

ω_k – Connectivity of host k, functions as a checkpointing node.

p_i – connectivity of host l, is functioning as a task executer.

φ_i – wireless link reliability.

Reliability driven middleware (ReD) are QoS aware as they are provided with the information via wireless measurements. As the distance increases the link reliability decreases, resulting to failures. In the algorithm proposed in this paper each provider gets a request message from a consumer which is in the proximity range of the provider. If the provider already has a dedicated consumer then gain of each relationship is checked. And a pair is formed of those two nodes having highest pairing gain amongst two relationships.

The pairing gain is calculated by: [12]

\[ \Delta \phi \approx (|| \phi_{rb} \times \phi_{bc} ||) / \Delta \phi \text{ alone} \]

Where,

\[ \phi \text{ - Connectivity of host C, ie, } \phi_{c} \text{ and } \phi_{b} \text{, and } \phi_{c} \text{, } \phi_{b} \text{, } \phi_{c} \text{, } \phi_{b} \text{, } \phi_{c} \text{, } \phi_{b} \text{, } \phi_{c} \text{, } \phi_{b} \text{, } \phi_{c} \text{, } \phi_{b} \text{, } \phi_{c} \text{, } \phi_{b} \text{, } \phi_{c} \text{, } \phi_{b} \text{.} \]

\[ \phi_{c} \text{ - Connectivity of node B, is serving as Checkpointing node.} \]

\[ \Delta t_i – Minimum checkpoint data transmission interval. \]

\[ \Delta t_{ic} = \Delta t_{bc} = \Delta t_{cb} \]

\[ \Delta t_{ic} – Minimum checkpoint data transmission time. \]

\[ \Delta t_{ib} – Minimum recovery data transmission time. \]

In [1] Parmeet Kumar and Pritee Parwekar has proposed a fuzzy rule based fault tolerance in mobile grids where the checkpointing arrangement is derived without accessing any wired network or a static host. In this paper weights for the nodes in the system are derived using a Fuzzy rule system (FRS), where the weights are availability of stable storage and battery power. If a node has sufficient storage for saving the checkpoint that particular node qualifies to be a checkpointing storage node (CSN). Furthermore the nodes which are assigned high weights by the FRS are classified as CSNs who will accept the checkpoints of neighboring nodes.

Stable storage and battery level together decide the fitness of fuzzy set. Memory capacity and crisp values in the range of (0%, 100%) are taken as the parameters of stable storage and battery power respectively. Rephrase weights for membership function of stable storage are used from [9], [10] and are assumed to be 1MB to 10MB from storage and checkpoint or recovery message is of size few bytes. The weights are calculated based on Fuzzy rule depicted in below Table 1.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Stable Storage</th>
<th>Battery Power</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>2</td>
<td>Low</td>
<td>Intermediate</td>
<td>Low</td>
</tr>
<tr>
<td>3</td>
<td>Low</td>
<td>High</td>
<td>Intermediate</td>
</tr>
<tr>
<td>4</td>
<td>Intermediate</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>5</td>
<td>Intermediate</td>
<td>Intermediate</td>
<td>Intermediate</td>
</tr>
<tr>
<td>6</td>
<td>Intermediate</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>7</td>
<td>High</td>
<td>Low</td>
<td>Intermediate</td>
</tr>
<tr>
<td>8</td>
<td>High</td>
<td>Intermediate</td>
<td>High</td>
</tr>
<tr>
<td>9</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>

The proposed protocol constructs a limited size rule base of size \( n^2 \) where \( n \) is number of values used for ruling. In this case those are High, Low and Intermediate. So it will have a rule size of \( 3^2 \).

2.2 Checkpointing Algorithms

In [21] an antecedent graph approach for checkpointing is proposed which Antecedent Graphs (AGs) are checkpointed onto a stable storage periodically so that the size of the graph piggybacked on the message does not reduce the efficiency. We see how the graphs are formed considering 3 mobile agents. In the beginning each agent is at the state \( S_A, S_B, S_C \) respectively. The intervals are determined by the messages that are received. When agent B receives message \( m_1 \) from C, the deterministic interval and AG of state interval \( S_B \) provides information about the previous condition.

Ex. Formation of AG of graph for agent A, message \( m_2 \) is received by agent A from agent B. A combines the antecedence graph received from B to its own graph for the formation of the event \( S_B \). B sends a message \( m_0 \) to C. After this, the message \( m_1 \) is received by B from C; with the difference of antecedence graph of C (if a message from C has been previously received). B combines the AG received from C to its own graph for the formation of the event \( S_B \). After this, the message \( m_1 \) is sent to B from C. After this, C receives the message \( m_3 \) from A, with the antecedence graph of A. C combines the antecedence graph received from A to its own graph for the formation of the event \( S_C \).

They have also used a parallel checkpointing algorithm where each checkpoint is given a unique sequence number, which increases monotonically. When a node sends a request it attaches checkpoint request and numeric weight value in parallel the dependent agents (DA) make AGs of events which occur during the checkpointing stage. However all the agents specified in the AG receives the inquiry message from checkpointing agent (CA). If DA agrees then CA sends the weight indicating positive response to starting agent (SA). If the SA receives such responses from all DAs then the nodes information is stored in stable storage by BA.

Jack Dongarra, Thomas Hertault and Yves Robert in [20] revisited a Double Checkpointing and proposed a Triple checkpointing algorithm. In double checkpointing to avoid crash of the whole application by a single failure local checkpoints are replicated. This algorithm states that once checkpointing is done locally and again the same checkpointing algorithm is performed.
is replicated remotely. In case of application failures the check pointed data may be recovered from two points, therefore if the node fails the data can be recovered from local or a paired node increasing the reliability. The non blocking algorithm is given below:

- Periodic checkpoints are taken, with period \[ P = \delta + \theta + \sigma \].

Where,

\( \delta \) – Period of checkpointing locally and no application work is performed.

\( \theta \) – Period of checkpointing remotely. Each node checkpoints remotely, that is it exchanges its checkpointed data with its paired node.

\( \sigma \) – Period of application executing at full speed.

The weight, which is the amount of checkpointing data, is given by \[ W = (\theta - \phi) + \sigma = P - \delta - \phi \].

Where,

\( \phi \) - is some communication overhead.

In triple checkpointing algorithm the nodes triples rather than pairs. It is more reliable when a fixed amount of memory is available for checkpointing. When the failure occurs the local checkpoint is lost and the checkpoint has to be recovered from its buddy. Triple checkpointing algorithm works just the same as double checkpointing but here we add another secondary node for checkpointing one more time. This reduces the overhead occurred in double checkpointing. Here 1st node checkpoints on primary node 2 and secondary node 3. Similarly node 2 has 2 buddies’ node 1 and node 3 and node 3 has again 2 buddies’ node 1 and node 2.

3. Findings

Here we find that in check pointing arrangements the reliability middleware is used to maintain the reliability of the check pointing arrangement where as in other paper fuzzy logic is used to do the same. Reliability middleware makes a stable pairs where as other algorithm makes the arrangement depending on fuzzy rules both are way better then Random Checkpoint arrangement (RCA) algorithm, shown below.

4. Conclusion

From above survey we can conclude that ReD middleware and fuzzy algorithms are efficient and reliable then the RCA. Also, the checkpointing algorithms which does not include the base agent are more reliable then the algorithms including the base agent.

5. Future Scope

In future we may merge these techniques and try to make a hybrid algorithm which provides a reliable checkpointing arrangement, which is decentralized and with low communicational overhead.

References


