COSC 6397
Big Data Analytics

Reliability

Edgar Gabriel
Spring 2015

Dependability

- Module reliability measures
  - MTTF: mean time to failure
  - FIT: failures in time
    \[ FIT = \frac{1}{MTTF} \]  \hspace{1cm} (14)
  - Often expressed as failures in 1,000,000,000 hours
  - MTTR: mean time to repair
  - MTBF: mean time between failures
    \[ MTBF = MTTF + MTTR \]  \hspace{1cm} (15)
- Module availability:
  \[ M_A = \frac{MTTF}{MTTF + MTTR} \]  \hspace{1cm} (16)
Dependability - example

- Assume a disk subsystem with the following components and MTTFs:
  - 10 disks, MTTF=1,000,000h
  - 1 SCSI controller, MTTF=500,000h
  - 1 power supply, MTTF=200,000h
  - 1 fan, MTTF=200,000h
  - 1 SCSI cable, MTTF=1,000,000h
- What is the MTTF of the entire system?

Dependability - example (II)

- Determine the sum of the failures in time of all components

\[ FIT_{system} = \frac{1}{1,000,000} + \frac{1}{500,000} + \frac{1}{200,000} + \frac{1}{200,000} + \frac{1}{1,000,000} \]

\[ = \frac{10 + 2 + 5 + 5 + 1}{1,000,000} = \frac{23}{1,000,000} = \frac{23,000}{1,000,000,000} \]

\[ \rightarrow MTTF_{system} = \frac{1}{FIT_{system}} = 43,500h \]
Dependability - example (III)

- What happens if we add a second power supply and we assume, that the MTTR of a power supply is 24 hours?
- Assumption: failures are not correlated
  - MTTF of the pair of power supplies is the mean time to failure of the overall system divided by the probability, that the redundant unit fails before the primary unit has been replaced
  - MTTF of the overall system:
    \[
    \text{FIT}_{\text{system}} = \frac{1}{\text{MTTF}_{\text{power}}} + \frac{1}{\text{MTTF}_{\text{power}}} = \frac{2}{\text{MTTF}_{\text{power}}} \\
    \text{MTTF}_{\text{system}} = \frac{1}{\text{FIT}_{\text{system}}} = \frac{\text{MTTF}_{\text{power}}}{2}
    \]

- Probability, that 1 unit fails within MTTR: \(\frac{\text{MTTR}}{\text{MTTF}_{\text{power}}}

\[
\text{MTTF}_{\text{pair}} = \frac{\frac{\text{MTTF}_{\text{power}}}{2}}{\text{MTTR}} = \frac{\text{MTTF}_{\text{power}}^2}{2\text{MTTR}} = \frac{200,000^2}{2 \times 24} \approx 830,000,000
\]
Dependability - example (III)

• More generally, if
  - power supply 1 has an MTTF of $MTTF_{power_1}$
  - power supply 2 has an MTTF of $MTTF_{power_2}$

  $FIT_{system} = 1/MTTF_{power_1} + 1/MTTF_{power_2}$

  $MTTF_{system} = 1/FIT_{system}$

  $MTTF_{pair} = MTTF_{system}/(MTTR/\min(MTTF_{power_1}, MTTF_{power_2}))$

Or if either $power_1$ or $power_2$ have been clearly declared to be the backup unit

$MTTF_{pair} = MTTF_{system}/(MTTR/MTTF_{power_{backup}})$

Replication based Reliability

• Creating multiple copies of a possibly mutating object (file, file system, process, data)
• Useful for
  - High availability
  - Reducing response time
  - Integrity
  - Access any replica/copy should be indistinguishable

• Data replication
  - Useful if analysis not time critical

• Process replication
  - useful if time to completion important
Replication based Reliability

- Non-trivial in communicating parallel applications
  - Messages could be on the fly while a process dies
  - Who communicates with whom?
    - E.g. process 0 sends a message to process 1
      - Does every copy of process 0 send a message to every copy of process 1
      - If Yes:
        » could be used for data verification
        » Explosion in the number of messages
      - If No: how to decide which copy of process 0 communicates with which copy of process 1

The Volpex Approach

Redundancy and/or independent checkpoint/restarts → multiple physical processes per logical process

Volpex Goals:
- Application progress tied to the fastest process replica(s)
- Seamless progress despite failures
- Minimum overhead of redundancy
VolpexMPI

- **MPI library for execution of parallel application on volatile nodes**
- **Key features:**
  - controlled redundancy: each MPI process can have multiple replicas
  - Receiver based direct communication between processes
  - Distributed sender logging
- **Prototype implementation supports ~40 MPI functions**
  - point-to-point operations (blocking and non-blocking)
  - collective operations
  - communicator management

Point-to-point communication

- **Goal:** efficient handling of multiple replicas for each MPI process
  - avoid sending each message to all replicas
- **Concept:**
  - receiver based communication model
    - sender buffers message locally
    - receiver contacts sender process requesting message
    - sequence numbers used for message matching in addition to the usual message envelope (tag, communicator, sender rank, recv rank)
  - no support for `MPI_ANY_SOURCE` as of today
Volpex MPI design

- Data transfer based on non-blocking sockets
  - supports timeout of messages and connection establishment
  - handling of failed processes
  - adding of new processes at runtime

- Sender buffer management:
  - circular buffer containing message envelopes and data
  - oldest log-entries are being overwritten
  - size of the circular buffer limits as of today ability to retrieve previous messages

Managing Replicated MPI processes

- Team based approach:
  - Processes are spawned in teams
  - Only in case of failure, processes from different team is contacted
  - Optimal for homogeneous environments
Influence of redundancy level

- Performance impact of executing one (x1), two (x2) and (x3) replicas of each process
- Normalized to the single redundancy VolpexMPI execution times

Influence of process failures

- Double redundancy
- Failing processes from both teams
- Normalized to the double redundancy execution times
The Target Selection Problem revisited

- Identifying best set of replicas
- Beneficial to connect to fastest replica
- Will make fast replica slow by making it handle more number of requests

Target Selection Algorithms

- RO: Response Order Algorithm
  - Request a message from all replicas of a given MPI rank
  - Target is selected based on response order of replicas
  - Regularly repeated during execution

- ERO: Extended Response Order Algorithm
  - Same preliminary steps as RO
  - Change to next (slower) target in the list if difference in newest sequence number for a particular message exceeds a given threshold
Target Selection Algorithms (II)

- Double redundancy tests on a heterogeneous configuration
  - fast nodes: Gigabit Ethernet, 2.2 GHz
  - slow nodes: Fast Ethernet, 1.0 GHz
- Initially, both teams contain processes on fast and slow nodes
- Each MPI rank has one fast and one slow process
- Normalized towards double redundancy numbers on GE

![Graph showing normalized execution time for 8 and 16 processes](image)

Alternative to Replication

- Encoding approaches allow to reduce the costs for obtaining reliability compared to replication
- Example: parity based protection in RAID 5 schemes

<table>
<thead>
<tr>
<th>Data Byte 1</th>
<th>Data Byte 2</th>
<th>Parity Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>01101010</td>
<td>11001001</td>
<td>10100011</td>
</tr>
</tbody>
</table>

- Recovery

<table>
<thead>
<tr>
<th>Data Byte 2</th>
<th>Parity Byte</th>
<th>Recovered Data Byte 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>11001001</td>
<td>10100011</td>
<td>01101010</td>
</tr>
</tbody>
</table>

UNIVERSITY of HOUSTON
A fault-tolerant parallel CG-solver

- Tightly coupled
- Can be used for all positive-definite, RSA-matrices in the Boeing-Harwell format
- Do a “backup” every $n$ iterations

- Can survive the failure of a single process
- Dedicate an additional process for holding data, which can be used during the recovery operation
- Work-communicator excludes the backup process

- For surviving $m$ process failures ($m < np$) you need $m$ additional processes

The backup procedure

- If your application shall survive one process failure at a time

<table>
<thead>
<tr>
<th>Rank 0</th>
<th>Rank 1</th>
<th>Rank 2</th>
<th>Rank 3</th>
<th>Rank 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>22</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>26</td>
</tr>
</tbody>
</table>

or

$$b_i = \sum_{j=1}^{np} v_i(j)$$

- Implementation: a single reduce operation for a vector
- Keep a copy of the vector $v$ which you used for the backup
The backup procedure

If your application shall survive two process failures

<table>
<thead>
<tr>
<th>Rank 0</th>
<th>Rank 1</th>
<th>Rank 2</th>
<th>Rank 3</th>
<th>Rank 4</th>
<th>Rank 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>22</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td></td>
<td>26</td>
</tr>
</tbody>
</table>

\[ x_{1,1} + x_{2,1} + x_{3,1} + x_{4,1} = x \]

with \( x \) determined as in the Reed-Solomon Algorithm

The recovery procedure

- Rebuild work-communicator
- Recover data
- Reset iteration counter
- On each process: copy backup of vector \( v \) into the current version