Fault tolerance for distributed systems

Oleg Batrashev
Aim of this presentation

- give overview of fault tolerance classification
- present pros and cons of different techniques
- discuss present and future of FT
Classification

Fault tolerance

- forward
  - hardware level
  - software level
- backward
  - checkpoint based
    - uncoordinated, coordinated, event-driven
  - log-based
    - pessimistic, optimistic, causal

Checkpoint mechanism

- system level
- application level
Forward FT

Duplication of different software and hardware components
- tripled hardware on Mars rovers
- doubled database for public access

Application areas
- Real-time systems (public server)
- Critical systems (nuclear plant)

Too expensive for scientific applications
Domino effect

- process $A$ sends message $m$ to process $B$
- process $B$ receives message $m$
- process $A$ fails, message is $m$ is “unsent”
- process $B$ must be rolled back to “unreceive” the message
  - it may cause another message to be unsent
  - and another process to be rolled back
- this may end up in *Domino effect*
Backward FT

- Checkpoint based - application state is periodically saved and restored on failure
  - no overhead during application normal execution
  - expensive if application state is large, floods communication channels when checkpointing
  - usually needs to restart all processes
- Log-based - logs of undeterministic events are saved and replayed on failure
  - large overhead during application normal execution in case of much communication
  - does not need to restart all processes
  - can log events to/from external systems
- Mixed
Checkpoint based

- snapshot of application state saved to stable storage
- when application fails all processes rollback to their last consistent snapshot

**uncoordinated** every process does local checkpoint whenever desires
  *domino effect*

**coordinated** all processes coordinate to make global checkpoint
  network speed has increased for the last 15 years

**event-driven** processes use some events (algorithm) to decide when to checkpoint
  algorithm can be very complicate
Log-based

- processes save undeterministic events
- when process fails it is restarted and saved events are used to replay

pessimistic makes sure no event is lost due to failure (waits until log event is saved) can be too expensive

optimistic events are saved asynchronously, in case of failure finds out which processes must be rolled back domino effect

causal analyzed to make less logging actions, but ensure there is no domino effect
Checkpoint mechanism

- **Application level** - programmer changes source code to support checkpointing
  - manual
  - automated (generated)
  - compiler-assisted
  
  portable to many platforms
  application specific
  may need much effort from programmer

- **System level**
  - OS internal
  - library based
  
  platform specific
  transparent to application
Existing solutions

Many solutions assume there is no communication during checkpointing

- BlueGene/L

All (except one) known MPI solutions use or extends one MPI implementation

- FT-MPI extends MPI specification and provides some routines to deal with failure
- MPICH-V family of extensions use different approaches
- LAM-MPI, OpenMPI uses BLCR to rollback/restart

New implementation for every MPI
Need to install, configure and keep up to date.
Not aware of application algorithm, saves everything
$C^3$ system

$\text{C}^3$: A System for Automating Application-Level Checkpointing of MPI Programs Greg Bronevetsky et al, Cornell University, 2004

- build on top of MPI, ie works with any implementation
- parses and changes application code for checkpoint/restore functionality
- new approach, just one prototype implementation

<table>
<thead>
<tr>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>RROMP Thin layer</td>
</tr>
<tr>
<td>MPI implementation</td>
</tr>
<tr>
<td>Network</td>
</tr>
</tbody>
</table>
First time my “Checkpointing” Google blog alert had emitted something valuable
May be interesting to GRID people and system administrators
The Berkeley/Stanford Recovery-Oriented Computing (ROC) Project
http://roc.cs.berkeley.edu/