Debugging Large-Scale Parallel Applications is Challenging

• Large systems will have millions of cores in near future
  – Increased difficulty for developing correct HPC applications
  – Traditional debuggers don’t perform well at this scale

• Faults come from various sources
  – Hardware: soft errors, physical degradation, design bugs
  – Software: coding bugs, misconfigurations
Developer Steps When Debugging a Parallel Application

Questions a developer has to answer when an application fails:

- Line of code?
- Code region?
- Parallel task that failed?
- When did it fail? (detect abnormal application phase)

• Need for tools to help developers find root cause quickly

AutomaDeD’s Error Detection Approach

Phase Annotation

Application

Task₁ Task₂ … Taskₙ

P³MPI Profiler

Model₁ Model₂ … Modelₙ

Clustering

(1) Abnormal Phases
(2) Abnormal Tasks
(3) Characteristic Transitions
Types of Behavioral Differences

Run 1
MPI Application Tasks
1 2 3 ... n
Spatial (between tasks)

Run 2
MPI Application Tasks
1 2 3 ... n
Temporal (between time points)

Run 3
MPI Application Tasks
1 2 3 ... n

Semi-Markov Models (SMM)

- Like a Markov model but with time between transitions
  - Nodes: application states
  - Edges: transitions from one state to another

Transition probability

Time spent in current state (before transition)
SMM Represents Task Control Flow

- States correspond to:
  - Calls to MPI routines
  - Code between MPI routines

Application Code

```c
main() {
    MPI_Init();
    ... Computation ...
    MPI_Send(..., 1, MPI_INTEGER, ...);
    for(...) foo();
    MPI_Recv(..., 1, MPI_INTEGER, ...);
    MPI_Finalize();
}

foo() {
    MPI_Send(..., 1024, MPI_DOUBLE, ...);
    ...Computation...
    MPI_Recv(..., 1024, MPI_DOUBLE, ...);
    ...Computation...
}
```

Semi-Markov Model

Two Approaches for Time Density Estimation: *Parametric and Non-parametric*

- Cheaper
- Lower Accuracy
- More Expensive
- Greater Accuracy
**AutomaDeD’s Error Detection Approach**

- **Phase Annotation**
  - Application
  - Tasks: Task$_1$, Task$_2$, ..., Task$_n$

- **P$^n$MPI Profiler**
  - Models: Model$_1$, Model$_2$, ..., Model$_n$
  - Clustering

- **Offline**
  - (1) Abnormal Phases
  - (2) Abnormal Tasks
  - (3) Characteristic Transitions

---

**User’s Phase Annotations**

Sample Code:

```c
main() {
    MPI_Init();
    // Computation...
    MPI_Send(..., MPI_INTEGER, ...);
    MPI_Pcontrol();
    for(...) {
        MPI_Send(..., MPI_DOUBLE, ...);
        // Computation...
        MPI_Recv(..., MPI_DOUBLE, ...);
    }
    MPI_Pcontrol();
    // Computation...
    MPI_Recv(..., MPI_INTEGER, ...);
    MPI_Finalize();
    MPI_Pcontrol();
}
```

- Phases denote dynamically repeated regions of execution
- Developers annotate phases in the code
  - MPI_Pcontrol is intercepted by wrapper library
A Semi-Markov Model per Task, per Phase

Task 1

Task 2

Task n

Phase 1 Phase 2 Phase 3 Phase 4

time

time

time

AutomaDeD’s Error Detection Approach

Offline

Phase Annotation

Application

Task_1 Task_2 ... Task_n

Online

P^N\text{MPI Profiler}

Model_1 Model_2 ... Model_n

Offline

Clustering

(1) Abnormal Phases
(2) Abnormal Tasks
(3) Characteristic Transitions
Faulty Phase Detection:  
*Find the Time Period of Abnormal Behavior*

- **Goal:** find phase that differs the most from other phases

**Sample runs available:**

Without sample runs:

**Clustering Tasks’ Models: Hierarchical Agglomerative Clustering (HAC)**

\[ \text{Diss}(\text{SMM}_i, \text{SMM}_j) = L2 \text{ Norm (Transition prob.)} + L2 \text{ Norm (Time prob.)} \]

Each task starts in its own cluster

- **Step 1**
  - Task 1 SMM
  - Task 2 SMM
  - Task 3 SMM
  - Task 4 SMM

- **Step 2**
  - Task 1 SMM
  - Task 2 SMM
  - Task 3 SMM
  - Task 4 SMM

- **Step 3**
  - Task 1 SMM
  - Task 2 SMM
  - Task 3 SMM
  - Task 4 SMM

- **Step 4**
  - ?

Do we stop? or, Do we get one cluster?

We need a *threshold* to decide when to stop
How To Select The Number Of Clusters

Option 1:
• User provides application’s natural cluster count $k$

Option 2:
• Use sample runs to compute clustering threshold $\tau$ that produces $k$ clusters
  – Use sample runs if available
  – Otherwise, compute $\tau$ from start of execution
  – Threshold based on highest increased in dissimilarity
• During real runs, cluster tasks using threshold $\tau$

Cluster Isolation Example

Cluster Isolation: *to separate buggy task in unusual cluster*

Master-Worker Application Example

Normal Execution

Buggy Execution

Bug in Task 9
Transition Isolation:
Erroneous Code Region Detection

• Method 1:
  – Find edge that distinguishes faulty cluster from the others
  – **Recall:** SMM dissimilarity is based on L2 norm of edge’s parameters

• Method 2:
  – Find *unusual individual edge*
  – Edge that takes unusual amount of time (compared to observed times)

Visualization of Results

[Diagram showing clusters and isolated transition]

Isolated transition (cluster 2)

Fault Injection

• NAS Parallel Benchmarks:
  – BT, CG, FT, MG, LU and SP
  – 16 tasks, Class A (input)

• 2000 injection experiments per application:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIN_LOOP</td>
<td>Local livelock/deadlock (delay 1,5, 10 sec)</td>
</tr>
<tr>
<td>INF_LOOP</td>
<td>Transient stall (infinite loop)</td>
</tr>
<tr>
<td>DROP_MESG</td>
<td>MPI message loss</td>
</tr>
<tr>
<td>REP_MESG</td>
<td>MPI message duplication</td>
</tr>
<tr>
<td>CPU_THR</td>
<td>CPU-intensive thread</td>
</tr>
<tr>
<td>MEM_THR</td>
<td>Memory-intensive thread</td>
</tr>
</tbody>
</table>
**Phase Detection Accuracy**

- ~90% for Loops and Message drops
- ~60% for Extra threads
  - *Training* = sample runs available
  - Training significantly better than no training
  - Histograms better than Gaussians

**Cluster Isolation Accuracy:**

*Isolating the abnormal task(s)*

- Results assume phase detected accurately
- Accuracy of *Cluster Isolation* highly variable

- Accuracy up to 90% for extra threads
- Poor detection elsewhere due to fault propagation: buggy task → normal task(s)
**Transition Isolation**

- Injected transition in top 5 candidates
  - Accuracy ~90% for loop faults
  - Highly variable for others
  - Less variable if event order information is used

---

**MVAPICH Bug**

- Job execution script failed to clean up at job end
  - MPI tasks executer (`mpirun`, version 0.9.9)
  - Left runaway processes on nodes

- Simulation:
  - Execute **BT** (affected application)
  - Run concurrently runaway applications (**LU**, **MG** or **SP**)
  - Runaway tasks interfere with normal **BT** execution
MVAPICH Bug Results: 
**SMMs Deviation Scores in Affected Application**

Affected application: BT benchmark
Interfering applications: SP, LU, MG benchmarks

![Graph showing SMM deviation scores for BT benchmark with SP, LU, and MG benchmarks.]

**Concluding Remarks**

- **Contributions:**
  - Novel way to model and compare parallel tasks’ behavior
  - Focuses debugging efforts on time period, tasks and code region where bug is first manifested
  - Accuracy up to ~90% for phase detection, cluster and transition isolation (delays and hangs)

- **Ongoing work:**
  - Scaling implementation to work on millions of tasks
  - Improving accuracy through different statistical models (e.g., Kernel Density Estimation, Gaussian Mixture Models)