AutomaDeD: Automata-Based Debugging for Dissimilar Parallel Tasks

Greg Bronevetsky, Ignacio Laguna, Saurabh Bagchi,
Bronis R. de Supinski, Dong H. Ahn, and Martin Schulz

DSN, 2010
Jun, 30th

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

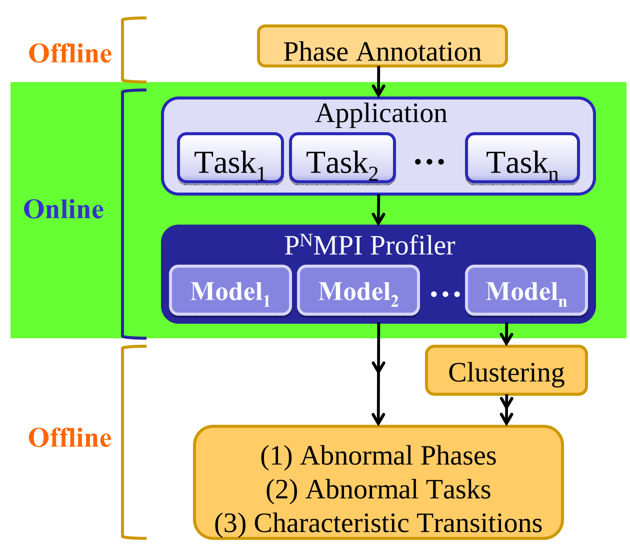
Questions a developer has to answer when an application fails:

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

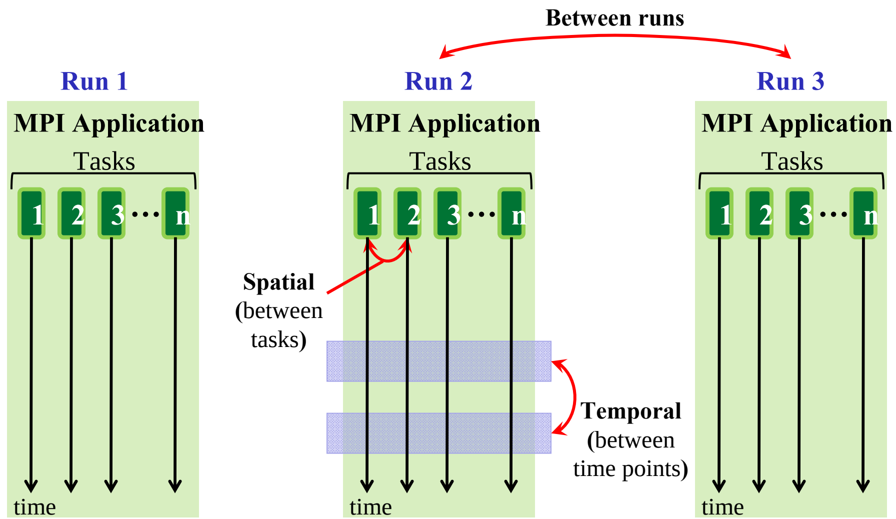
AutomaDeD

- Need for tools to help developers find root cause quickly

AutomaDeD's Error Detection Approach

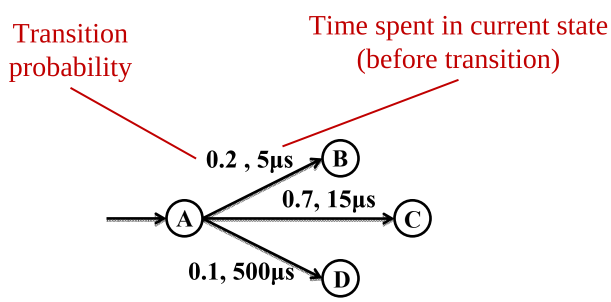


Types of Behavioral Differences



Semi-Markov Models (SMM)

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



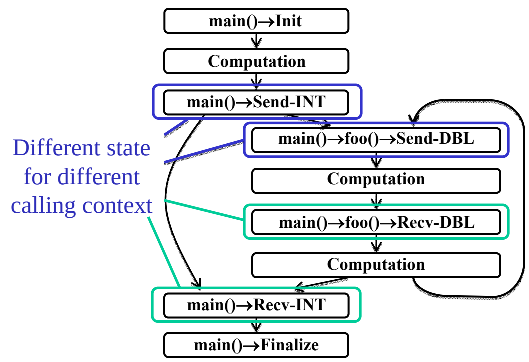
SMM Represents Task Control Flow

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

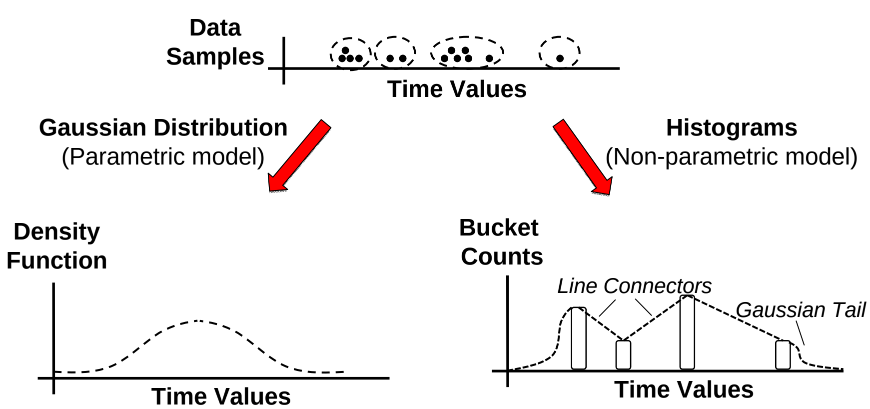
Application Code → Semi-Markov Model

```
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…
}
```



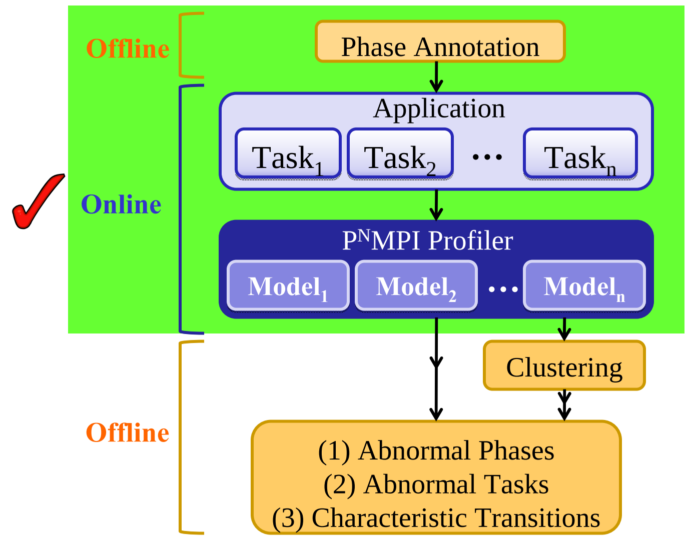
Two Approaches for Time Density Estimation: *Parametric and Non-parametric*



- Cheaper
- Lower Accuracy

- More Expensive
- Greater Accuracy

AutomaDeD's Error Detection Approach



User's Phase Annotations

Sample Code:

```
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();
}
```

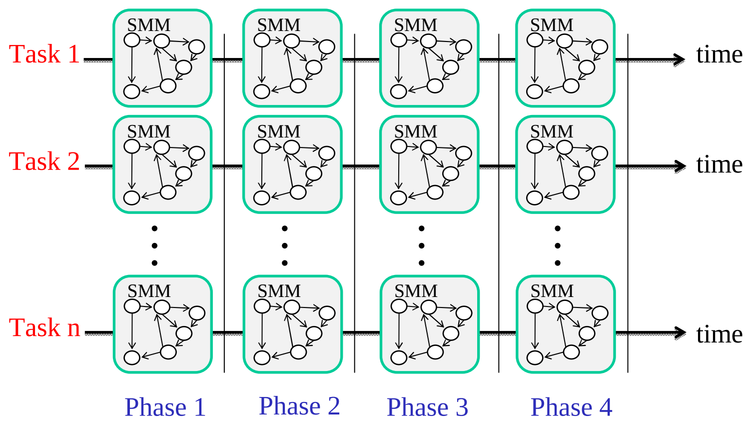
Phase 1

Phase 2

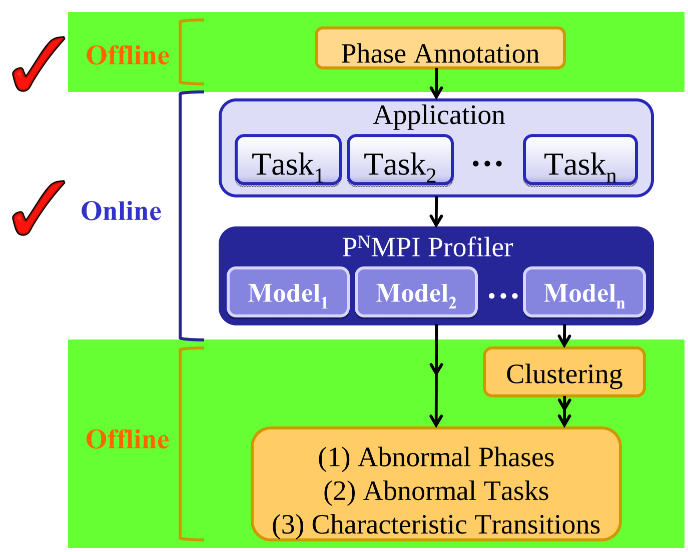
Phase 3

- 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



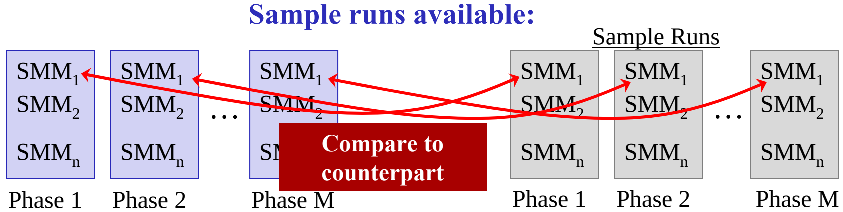
AutomaDeD's Error Detection Approach



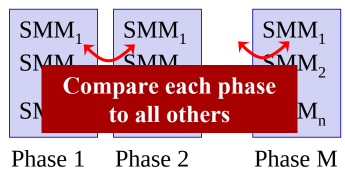
Faulty Phase Detection:
Find the Time Period of Abnormal Behavior

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

Sample runs available:



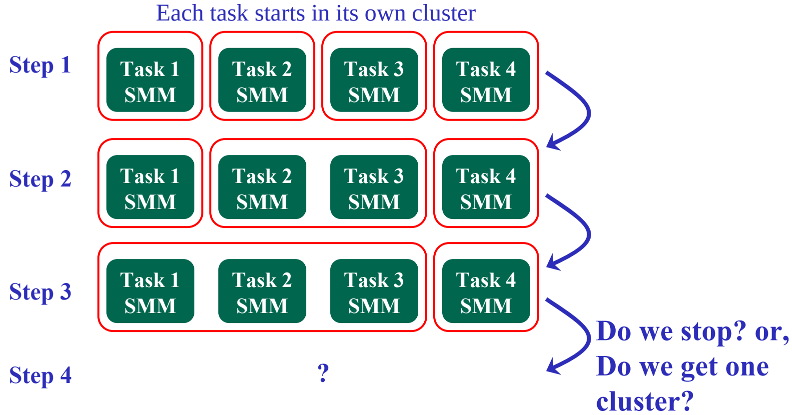
Without sample runs:



Clustering Tasks' Models:
Hierarchical Agglomerative Clustering (HAC)

Diss(SMM$_1$, SMM$_2$) = L2 Norm (Transition prob.) + L2 Norm (Time prob.)

Each task starts in its own 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**

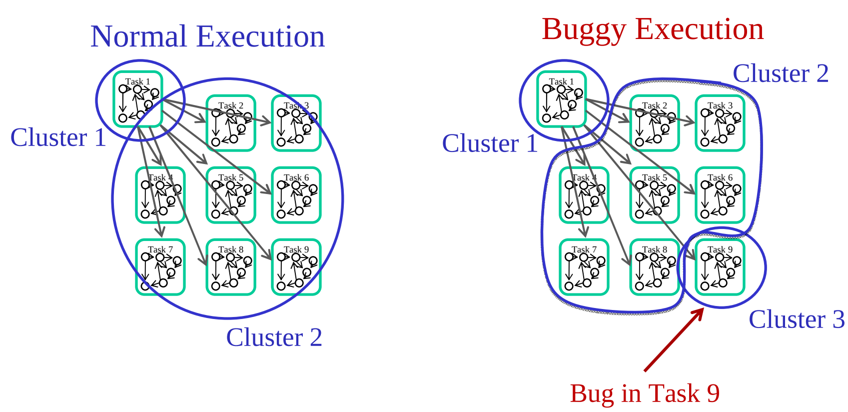
Option 2:

- Use sample runs to compute clustering threshold **τ** that produces **k** clusters
  - Use sample runs if available
  - Otherwise, compute **τ** from start of execution
  - Threshold based on highest increased in dissimilarity
- During real runs, cluster tasks using threshold **τ**

Cluster Isolation Example

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

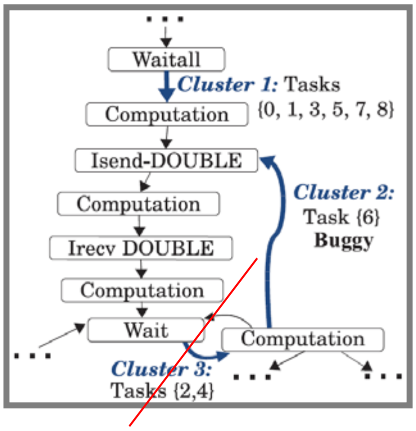
<u>Master-Worker Application Example</u>



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



Isolated transition (cluster 2)

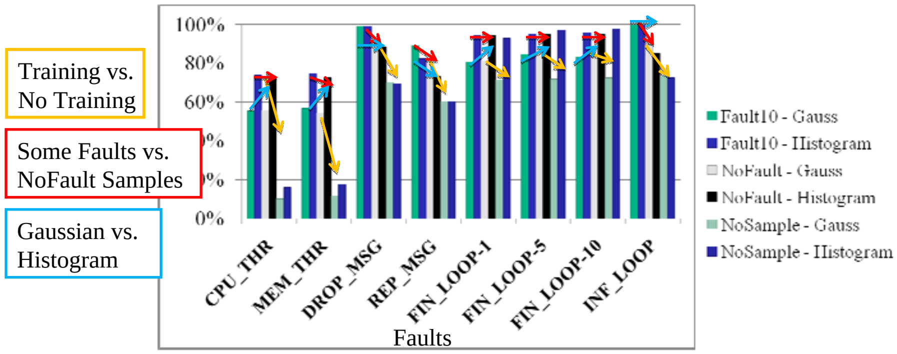
Fault Injection

- NAS Parallel Benchmarks:
  - BT, CG, FT, MG, LU and SP
  - 16 tasks, Class A (input)
- 2000 injection experiments per application:

Name	Description
FIN_LOOP	Local livelock/deadlock (delay 1,5, 10 sec)
INF_LOOP	Transient stall (infinite loop)
DROP_MESG	MPI message loss
REP_MESG	MPI message duplication
CPU_THR	CPU-intensive thread
MEM_THR	Memory-intensive thread

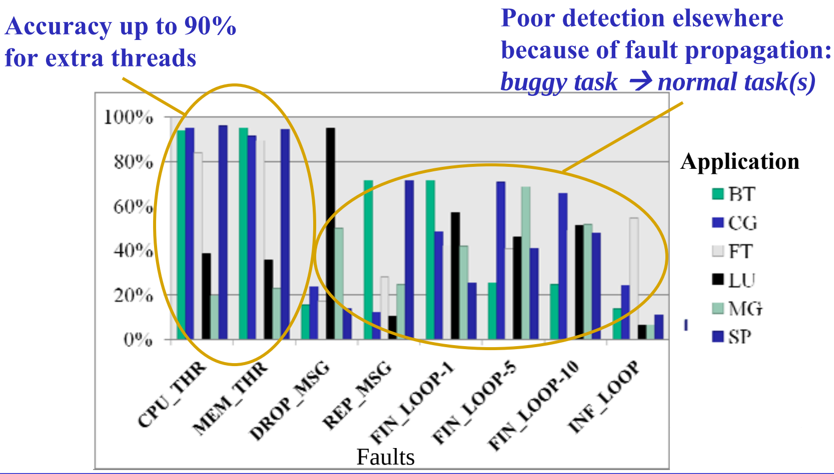
Phase Detection Accuracy

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



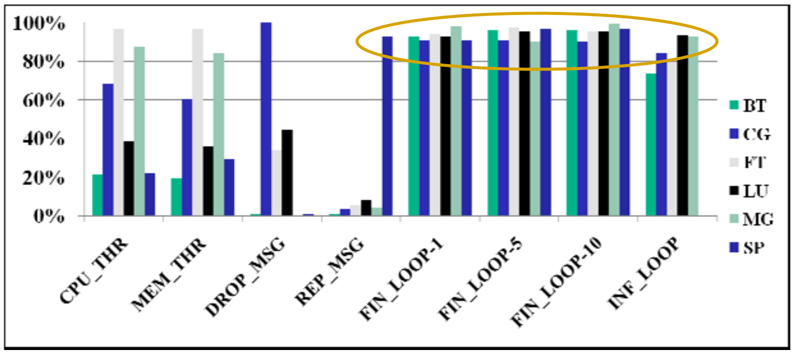
Cluster Isolation Accuracy: *Isolating the abnormal task(s)*

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



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

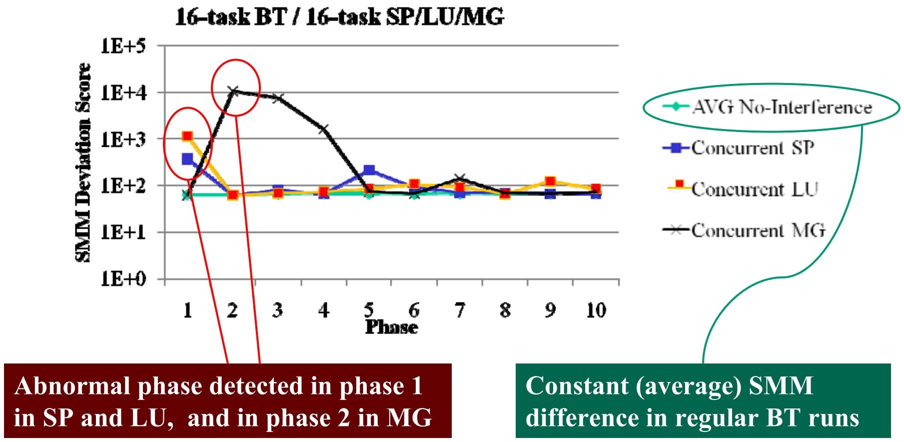


MVAPICH Bug

- 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



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)