Vrisha: Using Scaling Properties of Parallel Programs for Bug Detection and Localization

Bowen Zhou, Milind Kulkarni and Saurabh Bagchi
School of Electrical and Computer Engineering
Purdue University

PURDUE UNIVERSITY

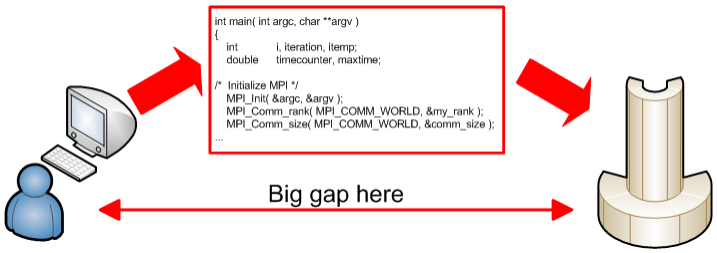
Scale-dependent Bugs in Parallel Programs

- What are they?
  - Bugs that arise often in large-scale runs while staying invisible in small-scale ones
  - Examples? Race condition, integer overflow, *etc.*
  - Severe impact on HPC systems: performance degradation, silent data corruption
- Difficult to detect, let alone localize

PURDUE UNIVERSITY

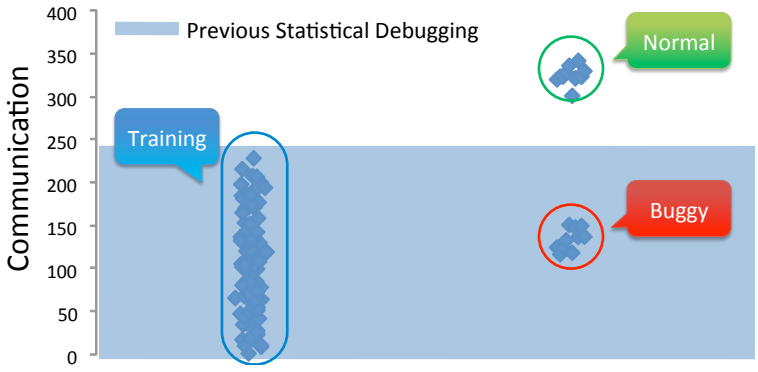
Scale-dependent Bugs in Parallel Programs

- Why do parallel programs have such bugs?
  - Coded and tested in small-scale development machines with small-scale test cases
  - Deployed in large-scale production systems



PURDUE UNIVERSITY

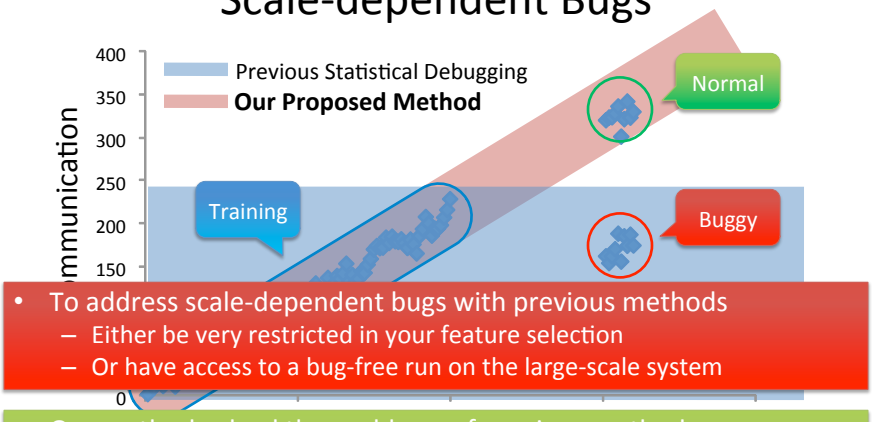
When Statistical Debugging Meets Scale-dependent Bugs



- Previous methods of statistical debugging for parallel programs
  [Mirgorodskiy SC'06] [Gao SC'07] [Kasick FAST'10]

PURDUE UNIVERSITY

When Statistical Debugging Meets Scale-dependent Bugs



- To address scale-dependent bugs with previous methods
  - Either be very restricted in your feature selection
  - Or have access to a bug-free run on the large-scale system
- Our method solved the problems of previous methods
  - Capable of modeling **scaling** behavior of parallel program
  - Only need access to bug-free runs from **small-scale** systems

PURDUE UNIVERSITY

Key Insights

- "**Natural scaling behavior**" of an application can be captured and used for bug detection
- Instead of looking for deviations from scale invariant behavior, we will look for deviations from the scaling trend

PURDUE UNIVERSITY

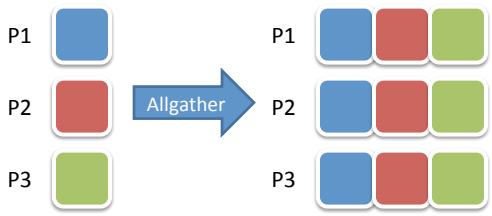
Contributions

- We build **Vrisha** to use scale of run as a parameter to statistical model for debugging scale-dependent bugs
- **Vrisha** is capable of building a model to deduce the correlation between scale of run and program behavior
- **Vrisha** detects both application and library bugs with low overhead as validated with two real bugs from a popular MPI implementation

PURDUE UNIVERSITY

Example of Scale-dependent Bugs

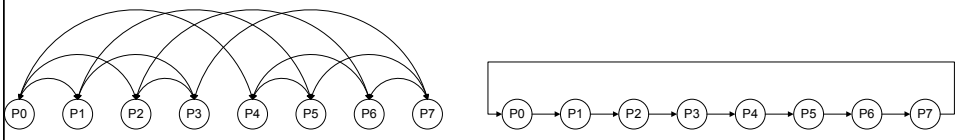
- A bug in MPI_Allgather in MPICH2-1.1
  - Allgather is a collective communication which lets every process gather data from all participating processes



PURDUE UNIVERSITY

Example of Scale-dependent Bugs

- MPICH2 uses distinct algorithms to do Allgather in different situations
- Optimal algorithm is selected based on the total amount of data received by each process



RECURSIVE DOUBLING

RING

PURDUE UNIVERSITY

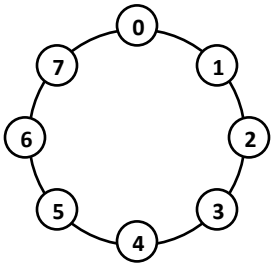
Example of Scale-dependent Bugs

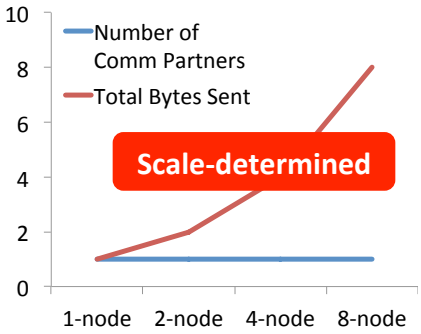
```
int MPIR_Allgather (
    ……
    int recvcount,
    MPI_Datatype recvtype,
    MPID_Comm *comm_ptr )
{
    int comm_size, rank;
    int curr_cnt, dst, type_size, left, right, jnext, comm_size_is_pof2;
    ……
    if ((recvcount*comm_size*type_size < MPIR_ALLGATHER_LONG_MSG) &&
        (comm_size_is_pof2 == 1)) {
        /* short or medium size message and power-of-two no. of processes.
         * Use recursive doubling algorithm */
    ……
    else if (recvcount*comm_size*type_size < MPIR_ALLGATHER_SHORT_MSG) {
        /* Short message and non-power-of-two no. of processes. Use
         * Bruck algorithm (see description above). */
    ……
    else {  /* long message or medium-size message and non-power-of-two
             * no. of processes. use ring algorithm. */
    ……
```

recvcount*comm_size*type_size can easily overflow a 32-bit integer on large systems and fail the if statement

PURDUE UNIVERSITY

Key Observation

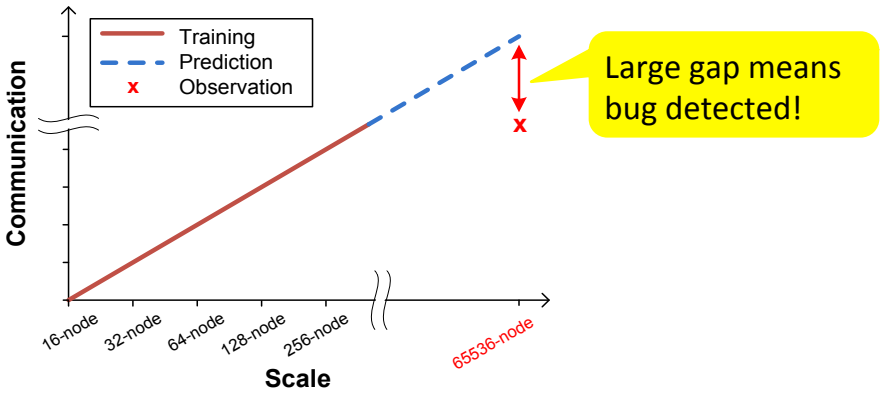




- We observe that some program properties are predictable by the scale of run in parallel programs
- These are called **scale-determined** properties

PURDUE UNIVERSITY

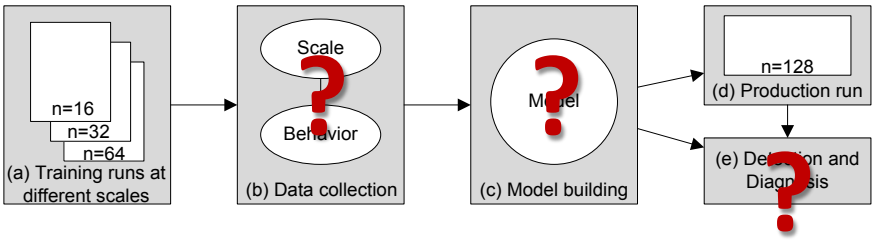
Bug Detection



Localization of bug is done by identifying the properties that cause the gap and the code regions that affect these properties

PURDUE UNIVERSITY

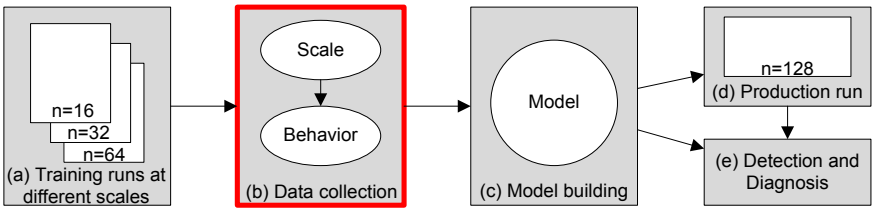
Vrisha's Workflow



a. Collect bug-free data from training runs of different scales
b. Aggregate data into scale and program property
c. Build a model from the scale and property
d. Collect data from the production run
e. Perform detection and diagnosis for the production run

PURDUE UNIVERSITY

Challenges



- What features should we use to build the model of scaling behavior?

PURDUE UNIVERSITY

Control Feature

- We generalize the concept of "scale" to **control features**
- A set of parameters given by system or user
  - number of processors in the system
  - command-line arguments
  - Input size
- Control features are the predictors of program behavior

PURDUE UNIVERSITY

Observational Feature

- To characterize program behavior, we use **observational features**
- A set of vantage points in source code to profile various runtime properties
- Observational features are the manifestations of program behavior

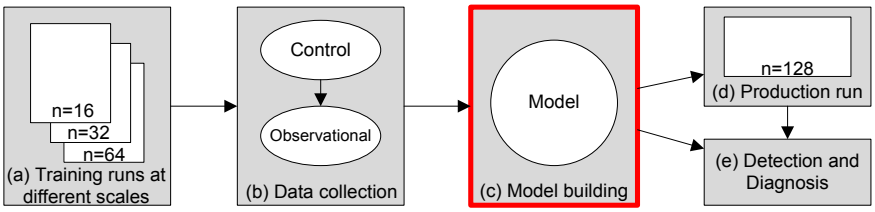
PURDUE UNIVERSITY

Observational Feature

- What does Vrisha use for observational feature?
  - Each unique call stack at the socket level under the MPI library as a distinct vantage point
  - Record the amount of data communicated at each vantage point as observational feature
- Why?
  - Capture both application and library bugs
  - Not need to modify application code
  - Impose low overhead
  - Provide source level diagnosis

PURDUE UNIVERSITY

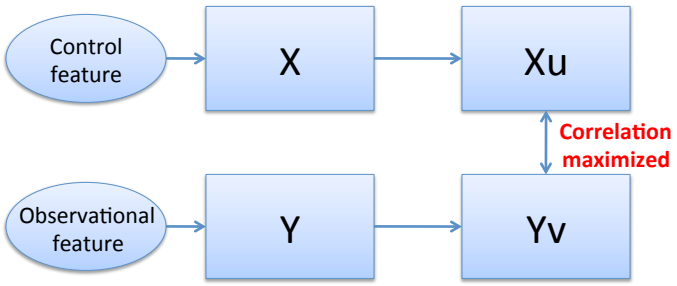
Challenges



- What model should we use to capture the relationship between scale and behavior?
  - Can describe both linear and non-linear relationships

PURDUE UNIVERSITY

Model: Canonical Correlation Analysis

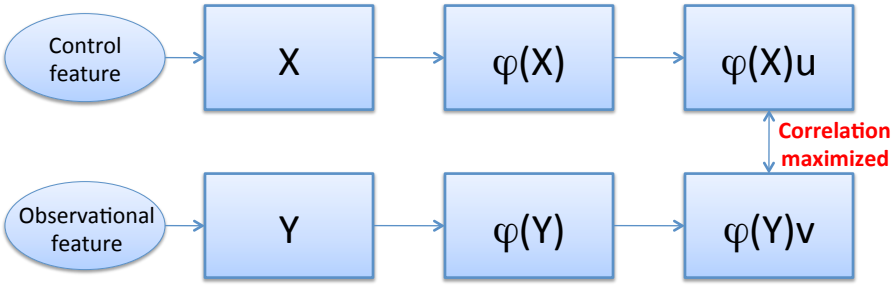


$$\max_{u,v} \quad corr(Xu, Yv)$$

$$\text{such that} \quad \|u\| = \|v\| = 1$$

PURDUE UNIVERSITY

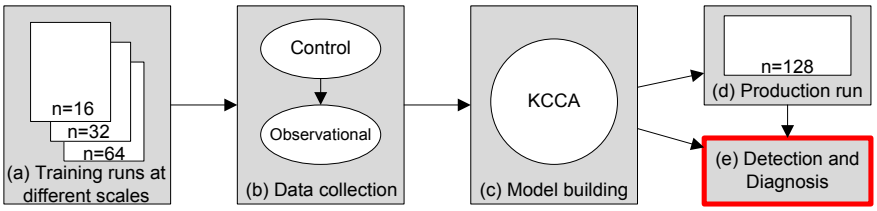
Model: Kernel CCA



We use "kernel trick", a popular machine learning technique, to transform non-linear relationship to linear ones via a non-linear mapping φ(.)

PURDUE UNIVERSITY

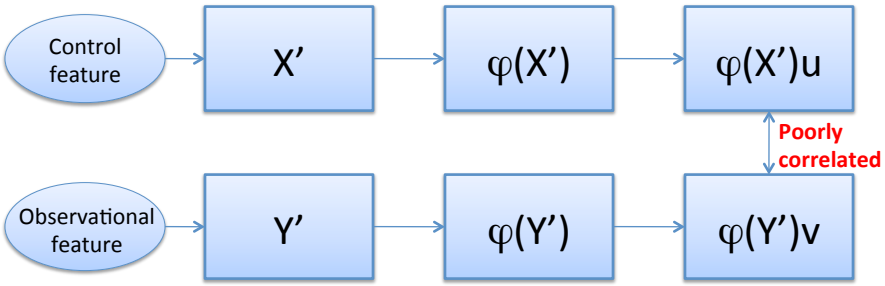
Challenges



- How does Vrisha do bug detection and diagnosis?

PURDUE UNIVERSITY

Bug Detection

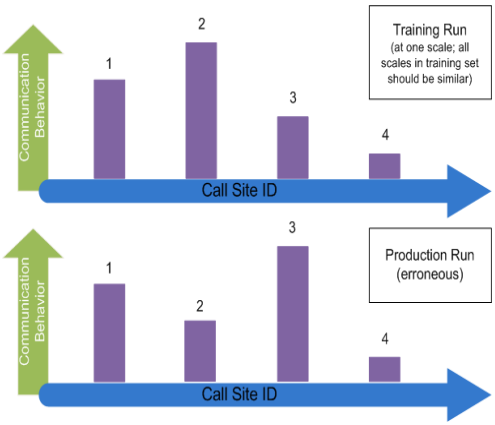


We declare a bug in the production run if the correlation between the control feature X' and the observational features Y' in the subspace defined by u and v is below certain range

PURDUE UNIVERSITY

Bug Localization

- Normalize communication volume of training and production runs to the same scale
- Comparing training (bug-free) with production (buggy) runs
- Identify the call stacks for the biggest changes as the potential location for bug



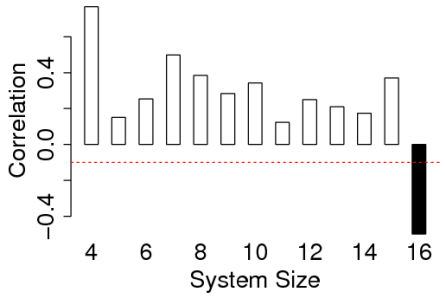
PURDUE UNIVERSITY

Evaluation

- We validated Vrisha with two real bugs from the MPICH2 library and Vrisha was able to detect and localize both of them
- Experiment setup
  - 16-node cluster
  - dual-socket 2.2GHz quad-core CPU
  - 512K L2, 8G RAM
  - MPICH2-1.1.1

PURDUE UNIVERSITY

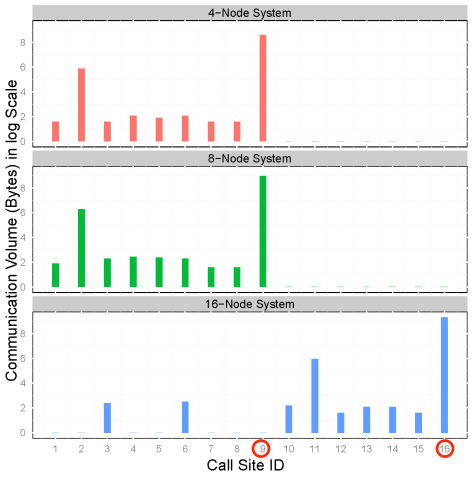
Detect the Bug in Allgather



- The bug is configured to be triggered at 16-node run only
- A KCCA model is built solely on 4- to 15-node runs
- Vrisha is capable of revealing this scale-dependent bug with the very low correlation in the 16-node buggy run

PURDUE UNIVERSITY

Locate the Bug in Allgather



- All the training runs like 4- and 8-node runs share the same pattern
- The 16-node run shows a different pattern
- The most radical difference happens between feature #9 and #16 which leads us to find the program makes a wrong choice at the buggy `if` statement

PURDUE UNIVERSITY

Vrisha's Overhead

- We measured the average overhead of Vrisha with NAS Parallel Benchmarks
- Vrisha's instrumentation overhead is less than 8% execution time on average
- Vrisha takes less than 30 ms to build model and less than 5 ms to detect bug, equivalent to around 1/10000 of the running time of these benchmarks on average

PURDUE UNIVERSITY  

Conclusion

- Some program properties are scale-determined in parallel programs and can be exploited to detect and localize **scale-dependent** bugs
- We built a system called **Vrisha** leveraging this idea and successfully detected two real bugs from the MPICH2 library with low overhead

PURDUE UNIVERSITY