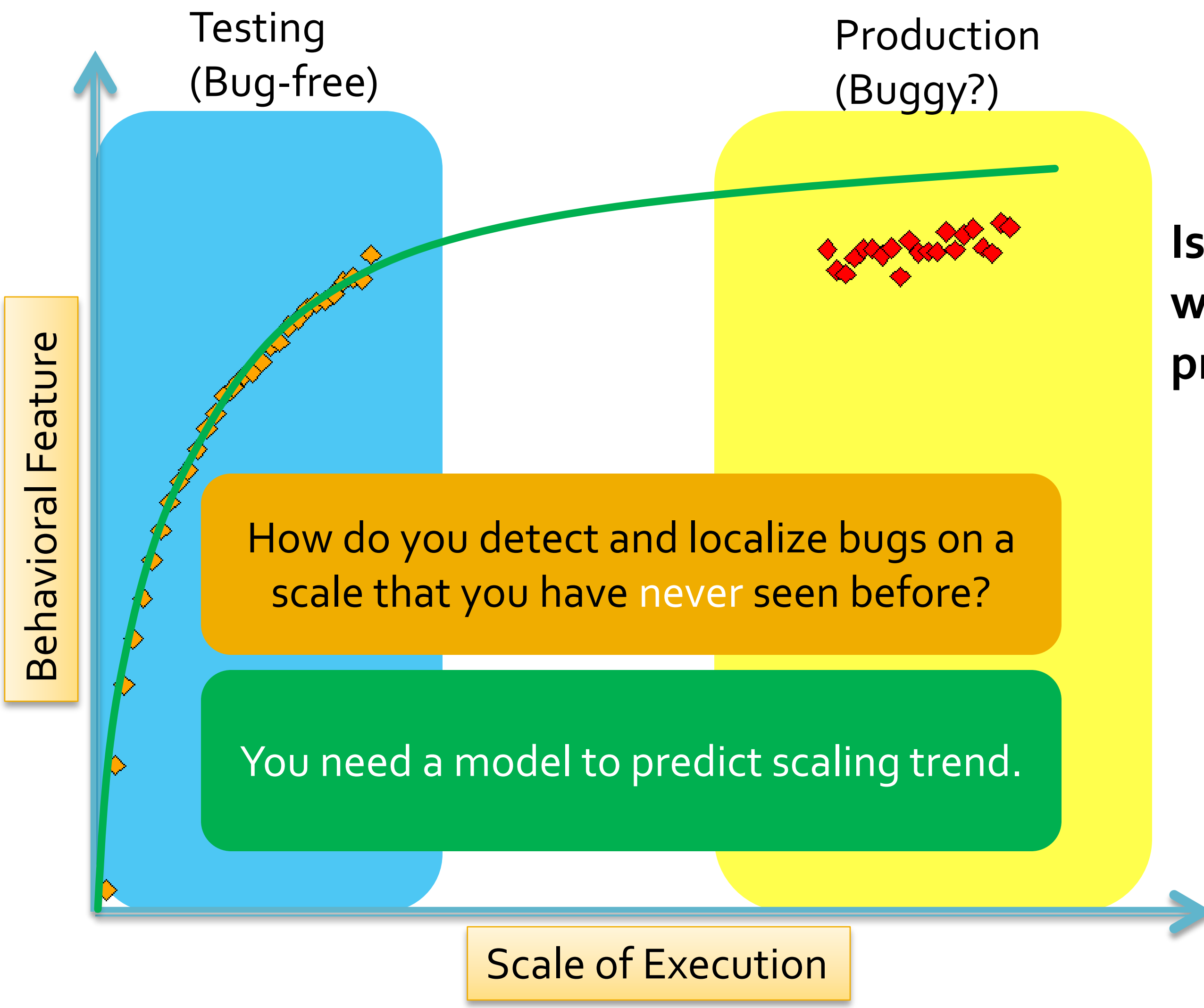
Bowen Zhou, Milind Kulkarni, and Saurabh Bagchi
Purdue University

ABHRANTA: Locating Bugs that Manifest at Large System Scales

Scale of Computing, Circa 2012

- Number of Processors
  - PC~8 cores
  - Workstation ~256 cores
  - Supercomputer ~1.5 mil cores
- Size of Data
  - Single Hard Drive ~4 TB
  - Hadoop HDFS ~21 PB
  - Lustre FS ~55 PB

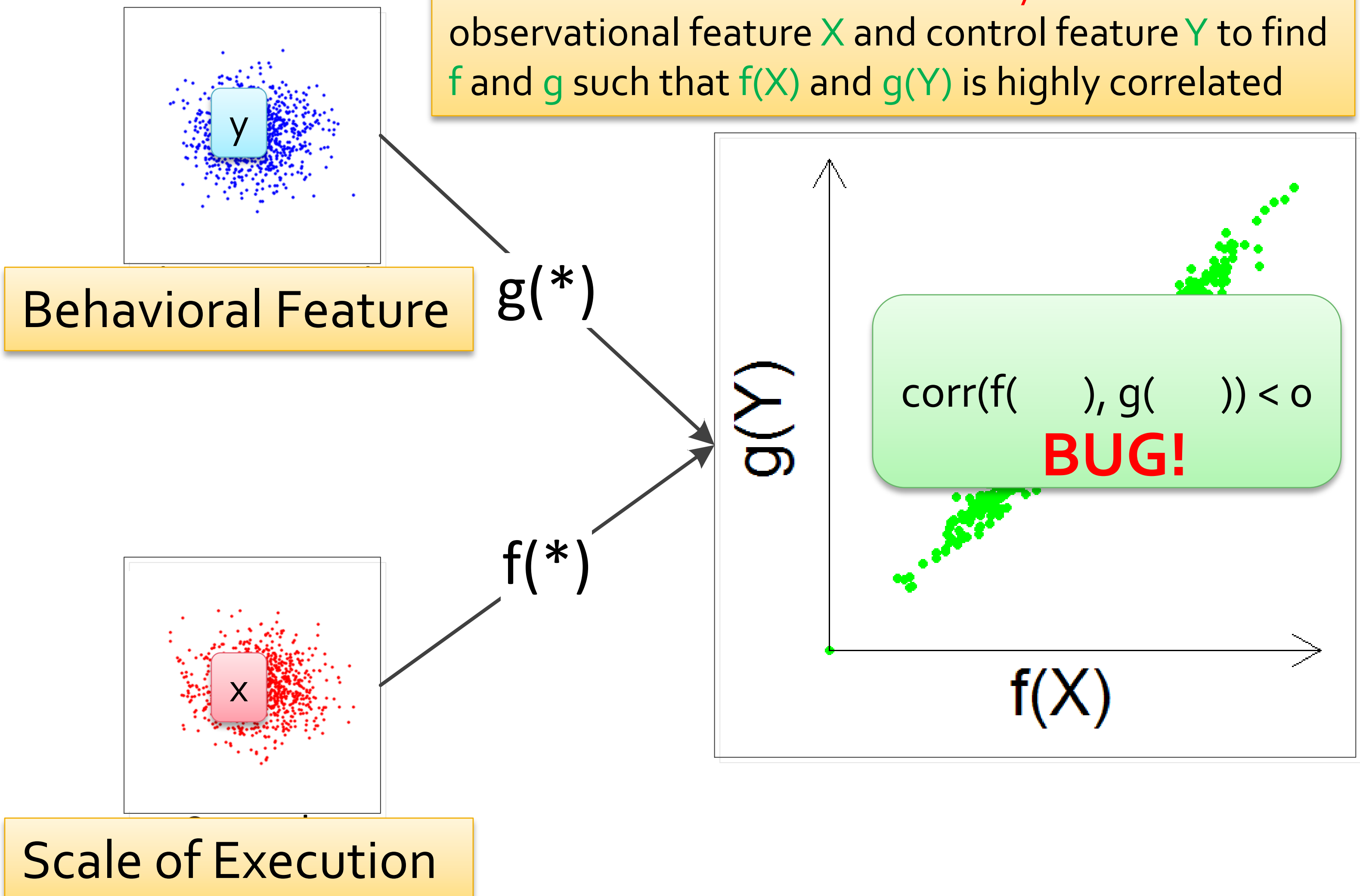
Scale-dependent Bug



Is there anything wrong in the production runs?

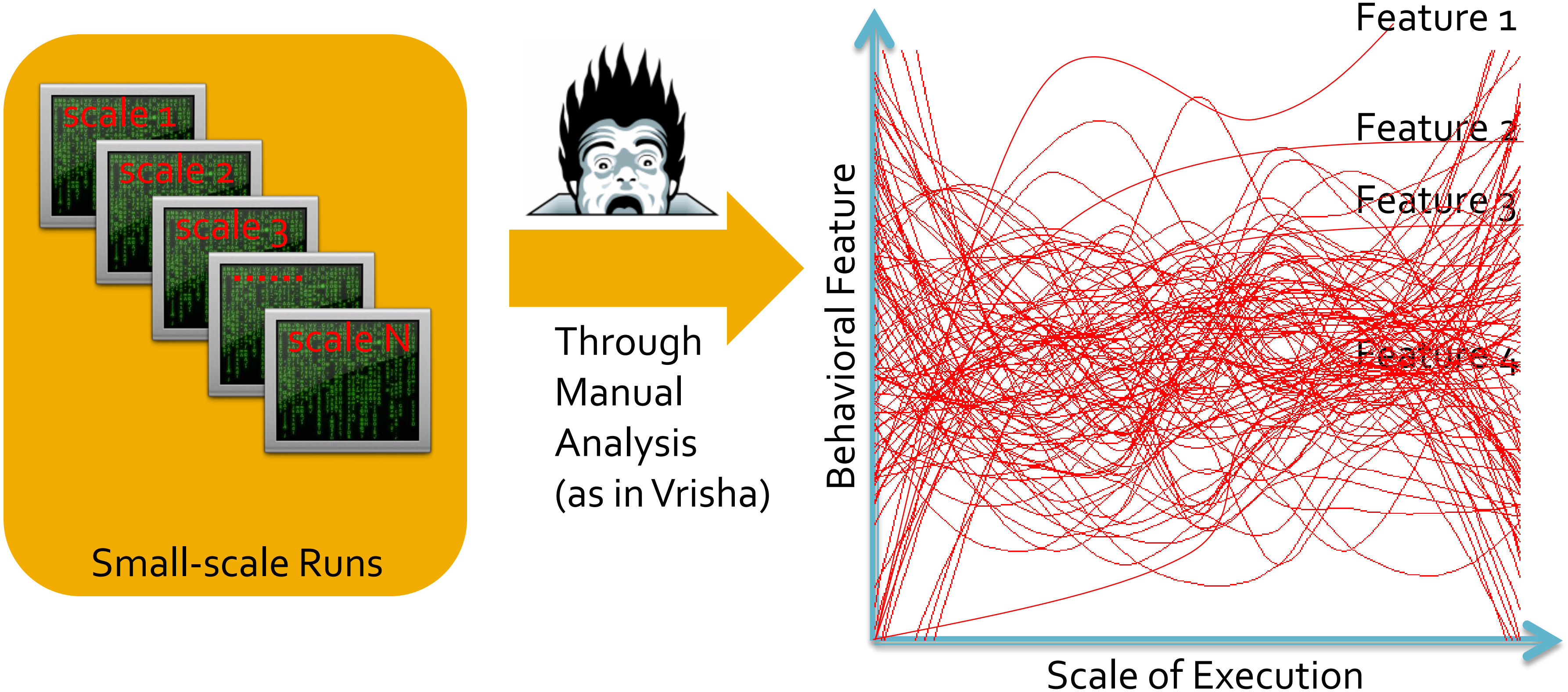
Vrisha: Using Scaling Properties for Bug Detection [HPDC '11]

Kernel Canonical Correlation Analysis takes observational feature X and control feature Y to find f and g such that f(X) and g(Y) is highly correlated

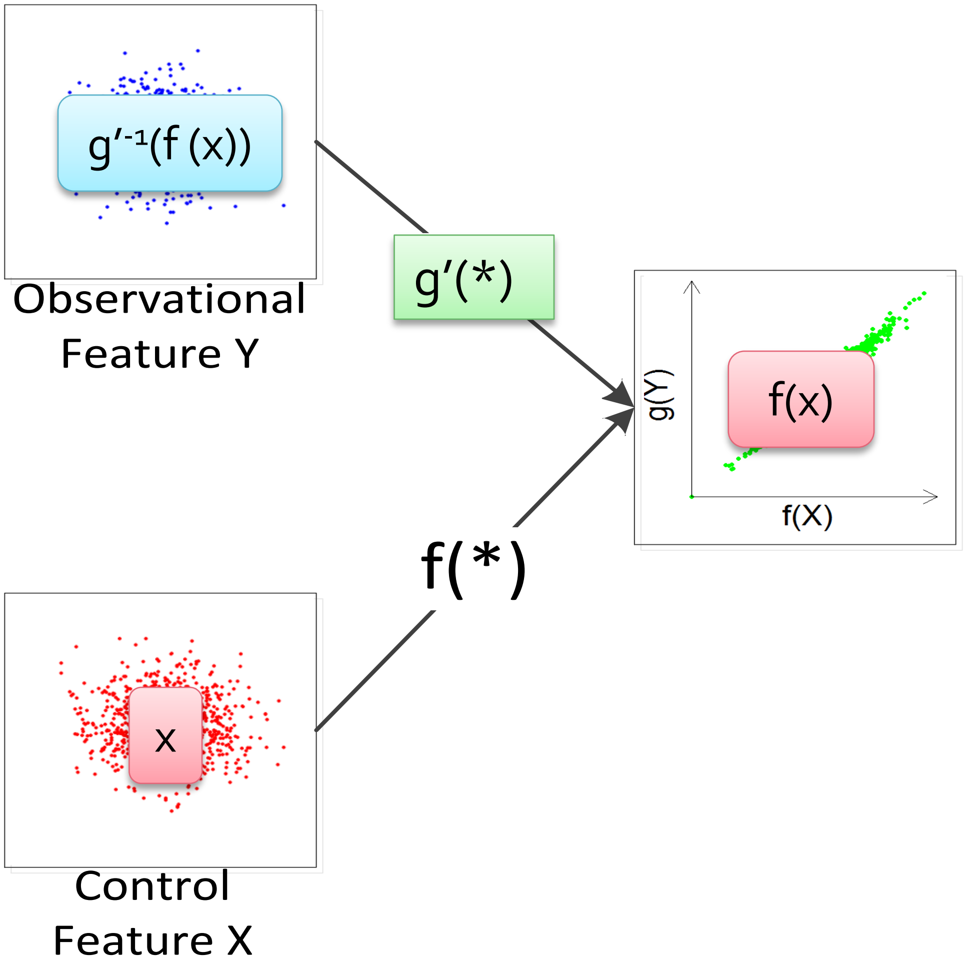


Vrisha: Bug Localization through Scaling Trend Extrapolation

- What is the "correct" behavior at large scale?
- Extrapolate large-scale behavior of each individual feature from a series of small-scale runs



ABHRANTA: a Predictive Model for Program Behavior at Large Scale



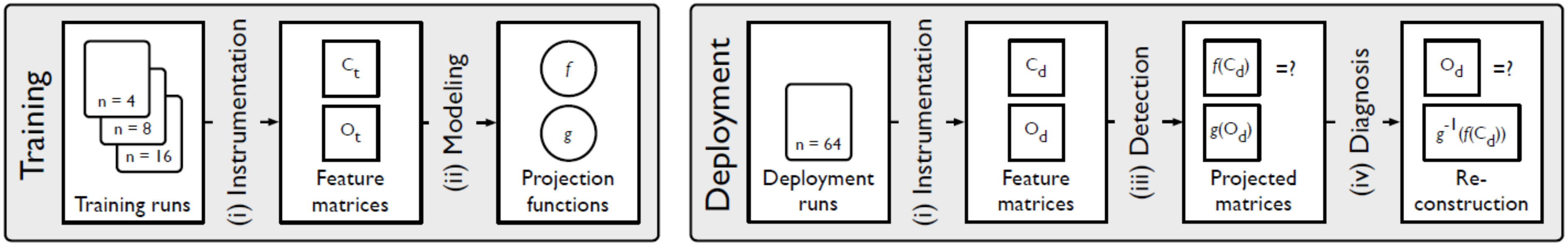
- ABHRANTA replaced non-invertible transform g used by Vrisha with a linear transform g′
- The new model provides an automatic way to reconstruct “bug-free” behavior at large scale, lifting the burden of manual analysis of program scaling behavior

ABHRANTA: Localize Bugs at Large Scale

- Bug localization at a large scale can be automated by contrasting the reconstructed bug-free behavior and the actual buggy behavior
- Identify the most "erroneous" features of program behavior by ranking all feature by:

$$|y - g'^{-1}(f(x))|$$

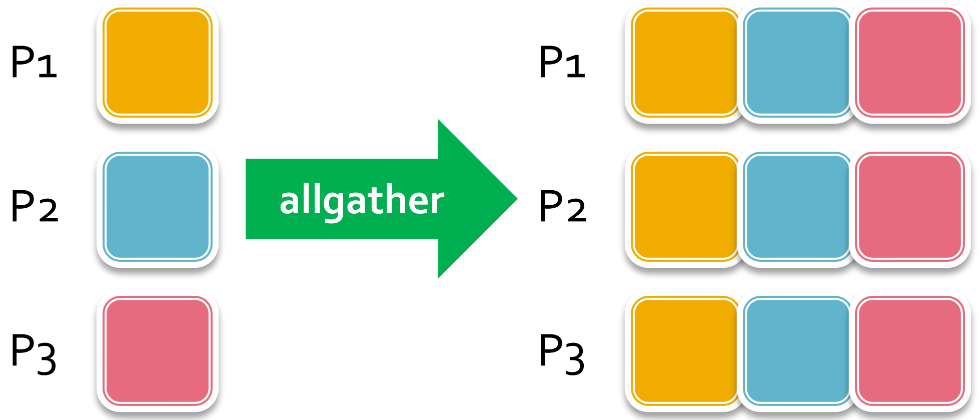
ABHRANTA's Workflow



- Training Phase (A Series of Small-scale Testing Runs)
  - **Instrumentation** to record observational features
  - **Modeling** to train a model that can predict observational features from control features
- Deployment Phase (Large-scale Production Runs)
  - **Instrumentation** to record the same features
  - **Detection** to flag production runs with negative correlation
  - **Localization**
    - Use the trained model to reconstruct observational feature
    - Rank features by reconstruction error

Case Study 1: Integer Overflow in MPICH2

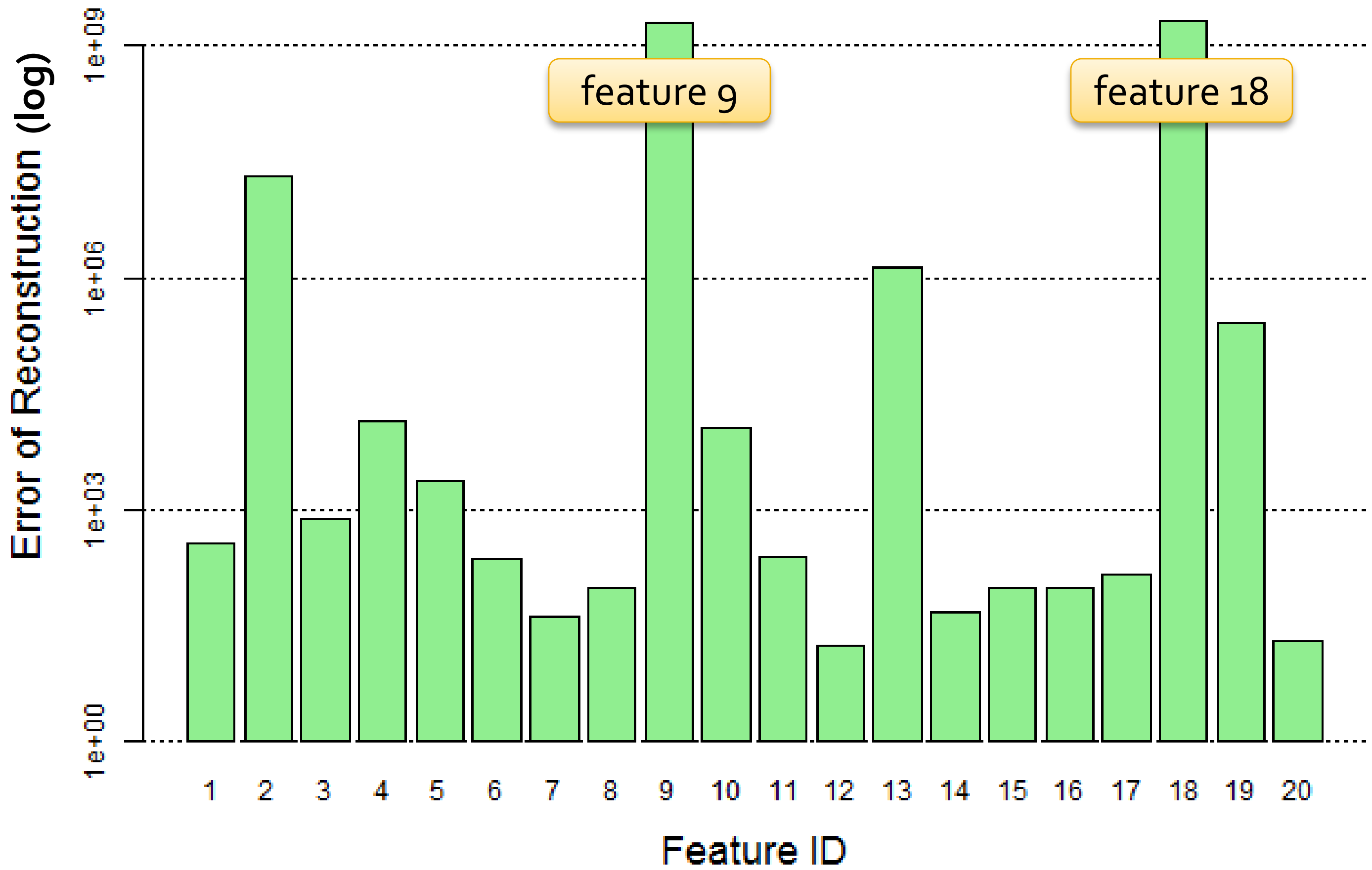
- allgather is an MPI function that allows a set of processes to exchange data with the rest of the group
- MPICH2 implemented 3 different algorithms to optimize the performance for different scales
- The integer overflow can make the function choose a suboptimal algorithm



Case Study 1: Integer Overflow in MPICH2

- Built a test application to trigger the bug at exactly 64 processes
  - Instrumented Socket API calls in MPICH2 with Pin
  - Control feature: the number of processes in a run and the rank of each process
  - Observational feature: the amount of data sent at every unique calling context of Socket API
- Trained the model with the data collected from 4- 15 process runs, localized the bug in a 64 process run

Case Study 1: Integer Overflow in MPICH2



Case Study 1: Integer Overflow in MPICH2

```
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)) {

        feature 18

    }
    else if (recvcount*comm_size*type_size < MPIR_ALLGATHER_SHORT_MSG) {

    }
    else {

        feature 9

    }

}
```

recvcount*comm_size*type_size can easily overflow a 32-bit integer on a large-scale run

Open Questions

- Feature selection
  - Correlated with scale
  - Related to the bug’s manifestation
- Non-deterministic behavior
  - Aggregate low-level features sharing the same prefix in their calling contexts
- Discontinuity in scaling trend
  - Require that the same scaling trend holds for all runs
- Generality
  - Verify with synthetic scale-dependent faults
  - Survey a large number of bugs that are scale-dependent

Conclusion

- We developed ABHRANTA, which leverages novel statistical modeling techniques to automate the detection and diagnosis of scale-dependent bugs
- With case studies of two real-world bugs, we showed that ABHRANTA is able to automatically and effectively diagnose bugs

Question?
bzhou@purdue.edu