Improving the Dependability of Data Center-Scale Computations
Daniel Porto*, Aniket Kate*, Rodrigo Rodrigues*, Pramod Bhatotia*, Alexander Wieder*
Flavio Junqueira†, Benjamin Reed†
Max Planck Institute for Software Systems (MPI-SWS)*, Yahoo! Research †

1. Context
• Increasing amount of data being processed in large data-centers
• At this scale, a variety of unlikely problems surface:
  - Network component failures
  - Undetected message corruption
  - Software bugs, arbitrary behavior
  - Storage component failures

The two broad classes of fault models are not well suited
Crash model: insufficient; does not cover arbitrary behavior
Byzantine model: considered overkill; internal data-center infrastructure is well protected against malicious adversaries
  - Ad hoc solutions have been applied (e.g., CRCs to protect state) – difficult to guarantee full coverage

2. Basic approach
• Find practical and systematic ways for handling non-crash (but non-adversarial) faults, by performing semantic checks

Given that checks are problem-specific, apply them to the infrastructure that supports large application base: Pig/Hadoop

Goals
• Deploy transparently to developer
• Perform lightweight checks trading cost for coverage to enable opportunistic verification
• Develop fault model to enable reasoning about effectiveness
• Generalize to other data-process systems

3. Developing semantic checks for Pig/Hadoop
Test whether various operators meets the following correctness requirements:
  - Integrity: output contains no corrupted or spurious records
  - Completeness: output contains all records that should be there

Goal is to develop effective techniques that check both properties while enabling trading cost for coverage

4. Example: Semantic check for Filter operator
• Build a set of verification objects (VOs) while scanning the data flowing into the operator
• Use these VOs to check for correctness while scanning data flowing out of the operator:

For input \(I_1, \ldots, I_n\) and output \(O_1, \ldots, O_m\), Filter check consists of:
  - Integrity: \(O_i \in I_1, \ldots, I_n\) and \(O_i\) meets filter condition
  - Completeness: either \(I_j \notin O_1, \ldots, O_m\) or \(I_j\) does not meet filter condition

Opportunistic verification
Trade cost for coverage by performing spot checking
Efficient inclusion tests might employ Bloom filters as a VO:
  - False positives in set inclusion imply a false negative in answer

5. Open challenges
How to amortize costly steps, e.g., Bloom filter creation?
• Attempt to merge checks for multiple operators, thereby only creating one Bloom filter per merged check

How can fault model capture less pessimistic assumptions?
• Consider fine-grained sub-computations (unit of fault)
• Assign a probability distribution to the faulty output
• Compute overall probability of checks failing

What guarantees are provided and are they sufficient?
How to handle extensibility mechanisms (user-defined functions)?

6. Final remarks
• Opens the opportunity to develop new ways to reason about non-crash faults
• Mechanism for handling non-crash faults in data center-scale computations
• Lightweight compared to Byzantine model
• Wide coverage of problems compared to crash model
• Tune cost for coverage according to resource availability