Introduction

- Massively data parallel multi-stage computations are difficult to verify.
- Growing access to parallelized data analysis/computation through recently emerging platforms [1, 2].
- Most of these computations are carried out on remote untrusted machines.
- Therefore, there is a need for verifying the correctness of these computations.

Our Approach

Graph based data flow model (largely inspired by Dryad [2]):
- vertices are sequential blocks adhering to the functional programming paradigm
- edges denote the data paths
To keep our verification mechanism as compatible as possible, we:
- place no restriction on the format of messages passed between the vertices
- don’t assume a priori knowledge of the computation itself.

Our verification scheme relies on random sampling and outputs probabilistic guarantees. We adapted the concept of "ringers" [3] to achieve a similar measure of confidence in the correctness of the computation.

Data Commitments

The input and output to each sampled node is verified using the composite pre- and post-commitments respectively (δ, and Δ).
- Each outgoing message is hashed and this hash output (δ) is stored with the message.
- λ is calculated by hashing over the set of all δ which correspond to the input messages.
- Λ is the root of the Merkle Hash Tree (MHT) which is formed by taking the δ hash values on the output side and using them as leaves of the MHT.
- λ and Λ serve a very similar purpose, yet have slightly different data structures.

Verifying Massively Data Parallel Multi-Stage Computations

Osama Khan and Randal Burns
Department of Computer Science, Johns Hopkins University

Algorithm

- At end of stage i, each node X should send SIGNX(X, λX, ΔX) to the auditor, where X ∈ {nodes in stage i}.
- Auditor chooses random set L ⊆ {nodes in stage i} of vertices for verification and for each vertex V ∈ L
  - Queries adversary for all input data M to V
  - Calculates λV = H(M) and verifies that λV == λX.
  - Queries adversary for witnesses needed to reconstruct each ΔY such that Y → V
  - Verifies that each reconstructed ΔY == ΔY (received during the verification of stage i-1)
  - Applies fY and generates all δ′Z such that V → Z
  - Computes Δ′Y and verifies Δ′Y == ΔY.

Analysis

Assuming that at each stage, ρ of the n_i vertices in stage i give corrupted results, then the probability of not detecting an anomaly at stage i is P(X_i = 0), where X_i follows the hypergeometric distribution.

P(X_i = 0) = \frac{(n_i - \mu_i)! \cdot (\mu_i - \rho_i)! \cdot \rho_i!}{n_i! \cdot (n_i - \rho_i)!}

Chaining these probabilities together gives us the probability of detecting an erroneous computation in s stages.

P(detection) = 1 - \prod_{i=1}^{s} P(X_i = 0)

Evaluation

The graph below shows a computation comprising 10 stages with 100 nodes each, 1% of the nodes randomly returning incorrect results, and only 5% of the nodes sampled per stage, giving us a 99% probability of detection.

This is explained by the fact that it is sufficient to detect an error at any stage to invalidate the entire computation. However, this independence of events is a double edged sword. If our random sampling strategy misses an anomaly at any stage, then it will definitely not be detected downstream.

References