Data parallel, multi-stage computations occurring on a massive scale are difficult to verify. With the recent advent of various parallel computing platforms [2, 4], and the resultant increased availability and accessibility of significant computational resources at a relatively economical price, more and more users are going to take advantage of the computational resources offered by the cloud [1]. It is also natural to assume that users are going to want some level of assurance about the correctness of the results of their computation. We attempt to form a generalized data flow model which captures the semantics of such massively parallel systems. We introduce a verification mechanism which is based on collecting commitments to the input and output data at each stage of the computation and then redundantly computing the results of a small subset computations in that stage. This verification mechanism is compatible with our generalized model and could be adapted to a variety of platforms, including Dryad and Map/Reduce.

Our scheme relies on random sampling, and outputs probabilistic guarantees. The type of computation can be completely arbitrary as our scheme does not rely on having any apriori knowledge of the function to be applied to the input data. Golle et al. [3] have used the idea of “ringers” as a random sampling technique to verify computations. We adapted it to achieve a similar measure of confidence in the correctness of the computations comprising an individual stage. While their adversarial model assumes an adversary motivated by financial gain, we do not place any such restriction. Our scheme also assumes the availability of a trusted auditing entity to carry out the actual verification.

Our data flow model is largely inspired by the one proposed for Dryad [4]. The user provides an acyclic graph where the vertices denote segments of sequential computation and the edges denote messages passed between the nodes. We do not impose any restriction on the format of these messages. Each message has associated metadata which allows us to trace its provenance as well as identify the node this message is destined to.

We verify each stage individually, which may be done interactively or as an offline batch verification task. After a particular stage finishes, the auditor solicits the commitments pertaining to the input and output data at all the nodes comprising that stage. It then redundantly computes the results of a random sample of nodes in that stage, computes their corresponding commitments and verifies that they match. Verification of a particular stage would build upon the successful verification of the immediately previous stage. It is straightforward to note that an anomaly detected at any stage is sufficient to raise an alarm, which allows us to improve the level of assurance as the number of stages increase. However, if our random sampling strategy misses an anomaly at a particular stage, then it will definitely not be detected downstream.

Preliminary results are encouraging. Fig. 1 shows a computation comprising 10 stages, with 1% of the nodes returning wrong results and 5% of the nodes sampled per stage, giving us a 99% probability of detection. Of course, this will vary with the particular corruption and sampling strategy deployed.

References


