Algorithm-Based Fault Tolerance Integrated With Replication

NASA’s Jet Propulsion Laboratory, Pasadena, California

In a proposed approach to programming and utilization of commercial off-the-shelf computing equipment, a combination of algorithm-based fault tolerance (ABFT) and replication would be utilized to obtain high degrees of fault tolerance without incurring excessive costs. The basic idea of the proposed approach is to integrate ABFT with replication such that the algorithmic portions of computations would be protected by ABFT, and the logical portions by replication.

ABFT is an extremely efficient, inexpensive, high-coverage technique for detecting and mitigating faults in computer systems used for algorithmic computations, but does not protect against errors in logical operations surrounding algorithms. Replication is a generally applicable, high-coverage technique for protecting general computations from faults, but is inefficient and costly because it requires additional computation time or additional computational circuitry (and, hence, additional mass and power). The goal of the proposed integration of ABFT with replication is to optimize the fault-tolerance aspect of the design of a computing system by using the less-efficient, more-expensive technique to protect only those computations that cannot be protected.
A common practice in computer science to associate a value with a key is to use a class of algorithms called a hash-table. These algorithms enable rapid storage and retrieval of values based upon a key. This approach assumes that many keys will need to be stored immediately. A new set of hash-table algorithms optimally uses system resources to ideally represent keys and values in memory such that the information can be stored and retrieved with a minimal amount of time and space. These hash-tables support the efficient addition of new entries. Also, for large data sets, the look-up time for large data-set searches is independent of the number of items stored, i.e., $O(1)$, provided that the chance of collision is low.

Like arrays, hash-tables provide constant time $O(1)$ look-up on average, regardless of the number of items in the table. However, the rare worst-case look-up time can be as bad as $O(n)$. Compared to other associative array data structures, hash-tables are most useful when large numbers of records are to be stored, especially if the size of the data set can be predicted.