# Technical Perspective: Taming Hardware Skew as Parallel DBMSs Scale Out

David J. DeWitt
Microsoft Corporation
Madison WI, U.S.A.

For almost 40 years now, relational database management systems have successfully used data parallelism to speed up the evaluation of large queries. Here, by "data parallelism" we mean taking one operation (for example, a "join" or an "aggregation") and spreading it over multiple machines, each operating on a part of the data. In general this approach works spectacularly well, yielding almost linear speedups over a wide variety of workloads. However, like any form of parallelism, data-parallel relational query processing is vulnerable to "skew." The database literature is full of work dealing with the skew that arises when one node in a parallel system is allocated more work than the average.

The following paper, by Li, Naughton, and Nehme, is interesting in that it deals with another kind of skew, one that has received much less attention: "hardware skew," that is, skew that arises because the processing units in a parallel system are not all of equal power. Such skew can arise in several ways – for example, a parallel system could be constructed "on the fly" by allocating available nodes in a cloud, or a company could upgrade an on-premises system with the addition of new nodes that are of a different generation and class of hardware than the existing ones. If the DBMS is oblivious to the fact that the underlying system is not uniform, the result will be the same as that achieved if the system were constructed entirely of the slowest nodes in the system.

If all the nodes in the system are equally "balanced" the solution is simple – if one node is 1/2 as fast as the average, give that node 1/2 the average work, and you are set. Unfortunately, in practice, things are not that simple. One node may have a faster CPU but the same I/O performance, or vice-versa; or nodes may have differing amounts of memory or network bandwidth. In such cases simple proportional allocation of work will be suboptimal. The situation is further complicated by the fact that different queries make different demands on the system with respect to CPU, memory, network, and disk; in fact, different stages of a single query can make very different demands.

This, finally, is the situation addressed by the paper, "Resource Bricolage for Parallel DBMSs on Heterogeneous Clusters." The authors make use of techniques for cost estimation growing out of the query optimization and query running time prediction literature; they combine these techniques with a linear programming model that chooses an optimal allocation for a given query on a given system. They demonstrate through an analytic model as well as experiments with an implementation that their proposed solution dominates simpler alternatives.

An interesting question this work raises is the duality between "on-demand" load balancing of the type employed by MapReduce-like systems and the predictive, up-front allocation of work advocated by this paper. My suspicion is that both approaches have their place, and the choice of which to use depends on issues such as the predictability of the workload and the importance of "locality" in the performance of the system. Perhaps hybrid solutions will be the answer in some cases.