Parallel Database Systems 101

Jim Gray,
310 Filbert St., San Francisco, CA. 94133
Gray@ crl.com

This talk is tutorial and remedial covering the basic ideas of parallel database systems. Other talks in this track go into detail on one or another product, so this talk just defines common terminology and ideas.

It begins by explaining why hardware trends (many inexpensive disks, drams, and microprocessors) force us to build parallel computers and program them in parallel. It describes the spectrum between shared-nothing, shared-disk, and shared memory systems and reports on recent progress in building scaleable shared memory systems and high-speed interconnects.

Parallelism trades money for time. Parallel systems are not cheaper, they are faster. Parallel systems must obey two laws: (1). The parallel system must be faster than a serial system, and (2): The parallel system must give near-linear speedup and scaleup.

Pipeline and Partition Parallelism are contrasted and the nemesis of blocking operators, startup, interference, and skew are defined. They limit speedup and scaleup.

OLTP and Decision Support (data warehouse) parallelism are contrasted. OLTP techniques and progress are briefly mentioned (TPC-A, B, C), but most of the talk focuses on batch (DSS) parallel query systems (TPC-D).

The recently approved TPC-D decision support database benchmark is reviewed. Published performance numbers will be given if they are available by then.

Database organizations (partitioning and indexing) will be reviewed. Utility requirements are reviewed (automatic, scaleable, parallel, online, incremental, restartable, monitored) with attention to data loading, reorganization, and recovery. Terabyte databases require online-everything as well as parallel-everything.

The presentation then turns to a whirlwind tour of parallelizing the relational operators: scan, select/project, sort, and join. The many approaches to join and indexing DSS data are reviewed (with special attention to join indices and bitmap techniques.)

The talk contains one small original contribution, a *benchmark buyers guide*. It proposes simple tests that can be applied to a parallel database system in an hour as a simple sanity check. These tests are simpler than the TPC-D benchmark that tests the query optimizer and the ability to run multiple-concurrent tasks.

The first test just sees if the system can scan data at disk speed. It starts with a million-record "Wisconsin" table (records are 208 bytes of 16 fields). The first test sees if the DBMS can scan the database at disk speed. Try the following query with parallelism turned off:

```
select count(*)
from wisc
where unique1 = unique2;
```

This query should run in less than a minute (about 3.4 MB/s or 17,000 records /sec). Pentiums run this fast, and they are the slowest computer one should buy.

Now grow the table to 4 million records spread across 4 disks and controllers. Repeat the query using 4 parallel processors (e.g., the 4-plex P6 systems). The resulting query should take less than a minute. If it takes longer, then there is something wrong with the system.

Now try the following query on the million-record Wisconsin table as a sequential query:

```
update wisc
set even_hundred = even_hundred + :zero;
```

This has to read and write each page of the table and generate a small log record for each one. This is a purely sequential operation but it is rare to find a system that can process this faster than 1MB/s (or 5,000 records per second). The systems are typically cpu bound. Now turn on parallelism and see if the query runs faster on multiple processors. You may want to partition the table into 4 pieces so that each processor has a separate partition. Ideally, 4 processors will update at 4 times the rate.

After the parallel update completes, call ROLLBACK WORK to see if transaction rollback is parallel as well. Often there are bottlenecks built into the transaction recovery logic.

Now try the sequence

```
delete wisc;
rollback work;
```

both as a sequential and 4-way parallel program. This sequence is likely to uncover bottlenecks in the transaction log mechanism and the transaction undo mechanism.

After all these updates are complete, fail the system and see if the restart time is comparable to the sequential processing time or the parallel processing time.

Surprisingly few systems pass these simple sanity tests.