TPC-B on Smallbase and P2[1]

Jeff Thomas and Don Batory
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712
{jthomas, batory}@cs.utexas.edu

August 7, 1995

1 Introduction

We ran a version of the TPC-B benchmark [TPC92] modified for main-memory DBMSs as described in [Hey94] on Smallbase [Hew95] and P2 [Bat94a-b]. The purpose of this experiment was to determine the relative performance of the current implementations of both systems.

Note that Smallbase and P2 do *not* currently provide the atomicity and durability properties of transactions. This experiment will help us to plan experiments and explain performance differences of future implementations of Smallbase and P2 that will provide these properties.

2 Procedure

The benchmark used Smallbase version 4.1, and P2 version 0.3. The benchmark was run on a HP 755 (99 MHz) with 128 MB of memory, and running HP-UX 9.01. Experiments were compiled using the HP-UX C compiler (**`/bin/cc`**) with the **`-O`** option. Experiments were run with relation size scale factor configurations of 2 through 12[2]. The database, relation, and index sizes/cardinalities were specified statically based on the scale factor. Each configuration was run 25 times, once each with random number seeds (the constant **`SeedVal`** in **`demo/tpcb.h`**) 1 through 25. Each run consisted of 200,000 (the constant **`NumXacts`** in **`demo/tpcb.h`**) non-atomic and non-durable "transactions". The results given are the median of the 25 runs for each scale factor configuration. The measure of performance we use is total transaction time (the variable **`totTime`** in **`demo/tpcbApiCT.c`**)

Each experiment used a different implementation of the data management functionality: (1) **`BASELINE`** removes all data manipulation functionality (i.e. all Smallbase and P2 calls are removed, so what remains is random number generation, statistics and timing maintenance, loop overhead, etc.), (2) **`SMALLBASE`** using the direct (rather than the SQL) interface (**`demo/tpcbApiCT.c`**) to the storage manager, and (3) P2 using various type equations.

1.This research was supported in part by Applied Research Laboratories at the University of Texas and Schlumberger.

2.The TPC-B benchmark requires that for each transaction per second (TPS) that a system claims to provide, there must be at least 100,000 **`account`** records, 10 **`teller`** records, and 1 **`branch`** record (a total of approximately 10MB). Thus, the relation size scale factor is required to be equivalent to TPS. Following [Hey94], however, we have de-coupled scale factor and TPS, because if the database were scaled as required (100,000 TPS * 10 MB/TPS = 1,000,000 MB), it would overflow main memory (128 MB).

The experiments included both **`persistent`** (or "permanent") and **`transient`** (or "temporary") databases. In **`BASELINE`**, the distinction between persistent and transient is irrelevant, since **`BASELINE`** does not actually store, retrieve, or manipulate any data. In Smallbase, persistent databases are specified by the **`sb_Perm`** and transient databases are specified by the **`sb_Temp`** argument to **`sb_dbCreate()`**. In P2, the distinction between transient and persistent (like all implementation choices) is specified using a type equation. The memory layer (**`mem_layer`**) is **`persistent`** for persistent databases and **`transient`** for transient databases. The **`persistent`** layer is implemented using the Unix system calls **`mmap()`** and **`munmap()`**.

`AVL` uses the type equation **`top2ds_qualify[avl[array[mem_layer]]]`**. This type equation describes a data manager that organizes tuples into an AVL-tree whose nodes are allocated from an array.

`QSORT` uses the type equation **`top2ds_qualify[qsort[mem_layer]]`**. This type equation describes a data manager that stores tuples in an array, orders the array using the quicker-sort algorithm, and performs retrievals using binary search in the ordered array.

`HASH0` and **`HASH1`** use the type equation **`top2ds_qualify[hash[array[mem_layer]]]`**. This type equation describes a data manager that stores tuples in a hash table which is an array of pointers to singly-linked lists of colliding tuples. **`HASH0`** uses the default P2 general-purpose hash function. **`HASH1`** uses a hash function—the identity function—that is customized for the TPC-B benchmark; besides being efficient to compute, the identity function avoids collisions, so the resulting hash table degenerates to an array of pointers to tuples.

`IARRAY` uses the type equation **`top2ds_qualify[iarray[mem_layer]]`**. This type equation describes a data manager that stores tuples in an array such that the index of a tuple is its key value. Each element in the array includes a flag indicating whether or not the element contains a valid tuple. This is probably the most efficient data structure possible for the TPC-B benchmark.

3 Results

Figure 1 presents the results for persistent databases and Figure 2 presents the results for transient databases. Note that for Smallbase, the TPC-B benchmark runs at *exactly* the same speed on both temporary and permanent databases. That is, the **`sb_Temp`** and **`sb_Perm`** arguments to **`sb_dbCreate`** do not change the performance of the resulting database. For P2, the TPC-B benchmark runs approximately 10 percent faster on transient databases than the corresponding persistent databases. Figure 3 summarizes the results for persistent versus transient databases.

Overall, the TPC-B benchmark ran faster on P2 than on Smallbase, and this performance advantage of P2 over Smallbase increased with the scale factor. Speedups for P2 over Smallbase ranged from 1.5 to 5, depending on the scale factor and P2 type expression.

4 Conclusions

These results reinforce our belief in the importance of being able to easily modify the implementation of a DBMS in order to match the needs of an application [Bat95]. In particular, these results show the importance of being able to match the indexing strategy to the application.

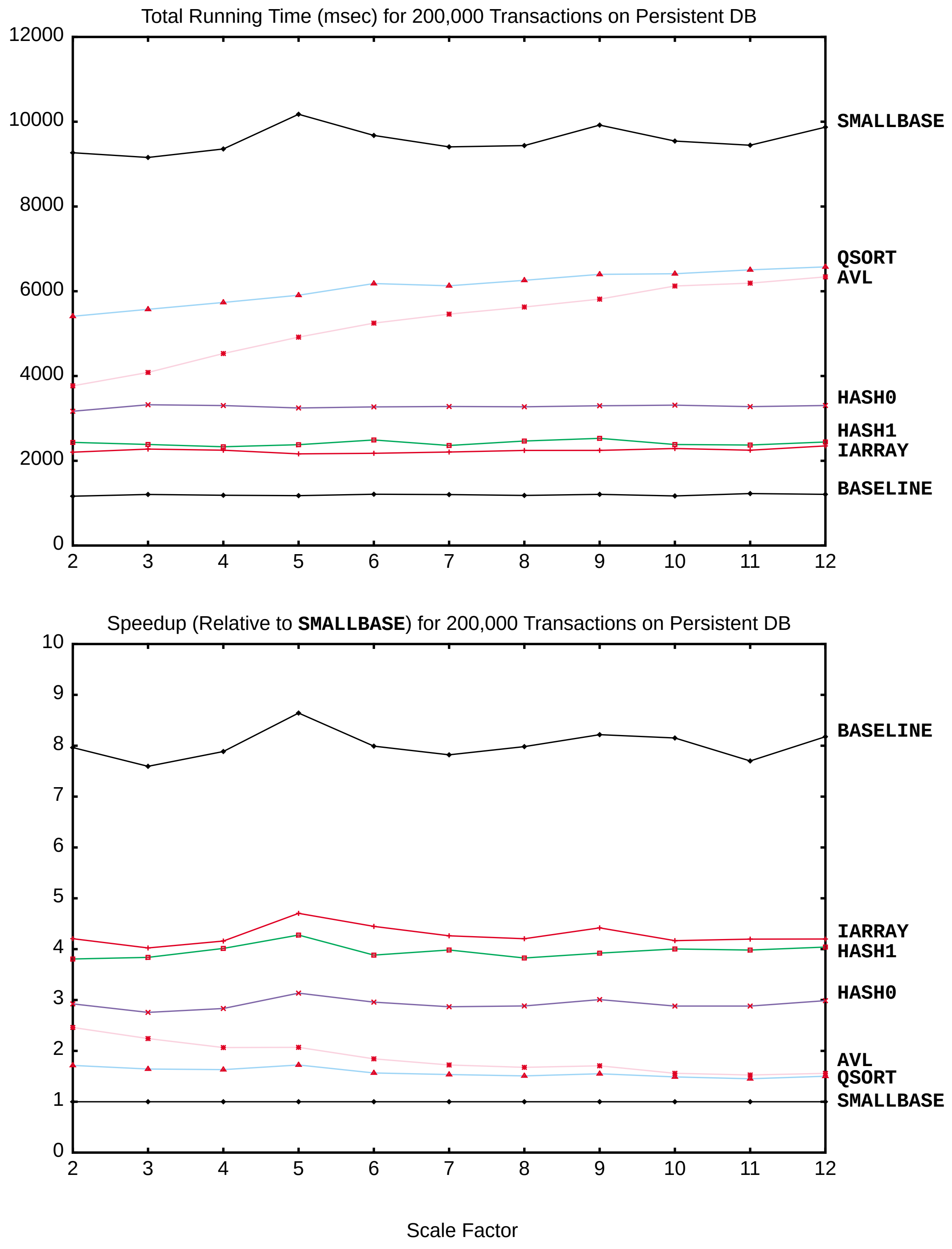


Figure 1: Total running time (msec) and speedup (relative to **SMALLBASE**) versus scale factor (relation size) for 200,000 transactions on persistent databases.

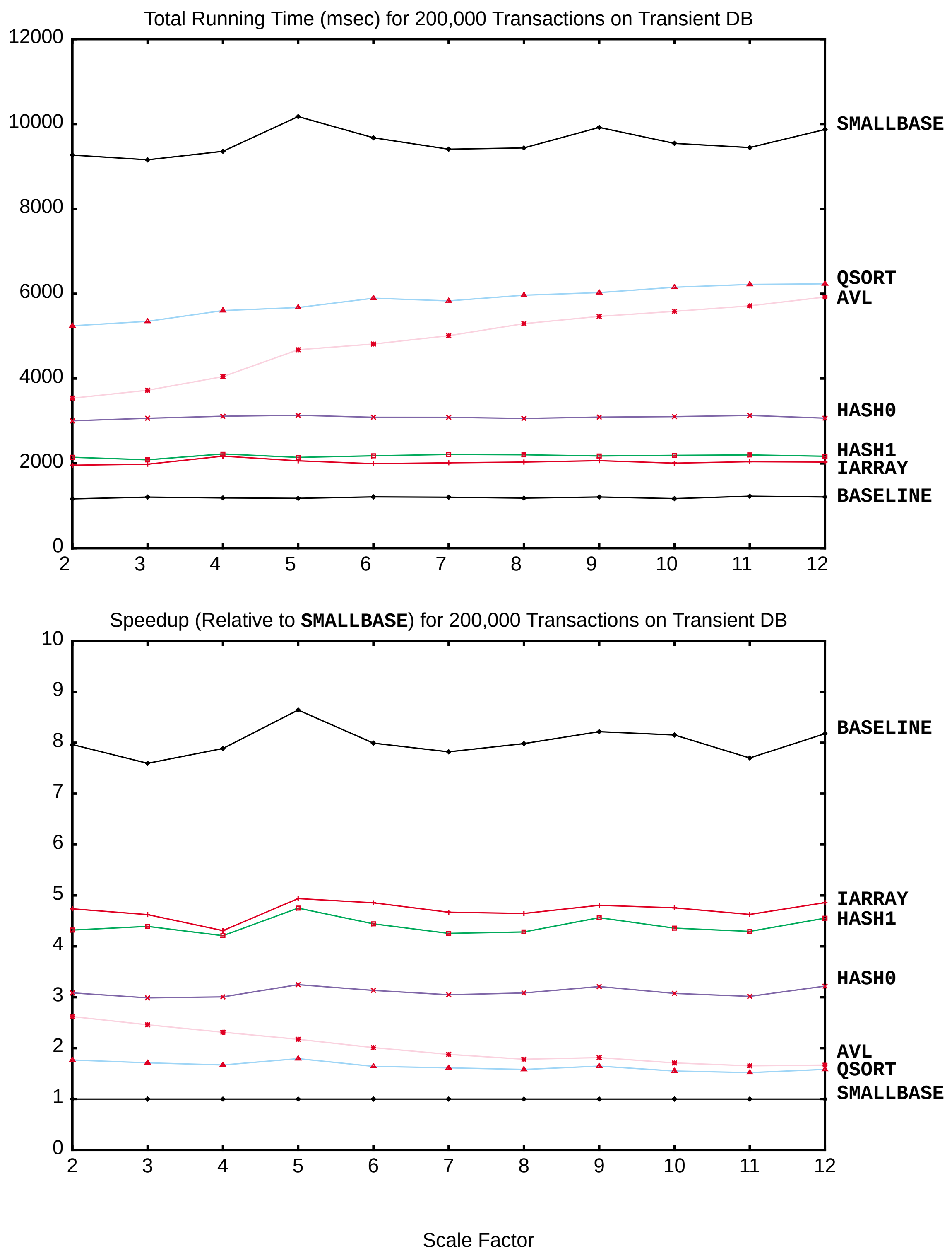
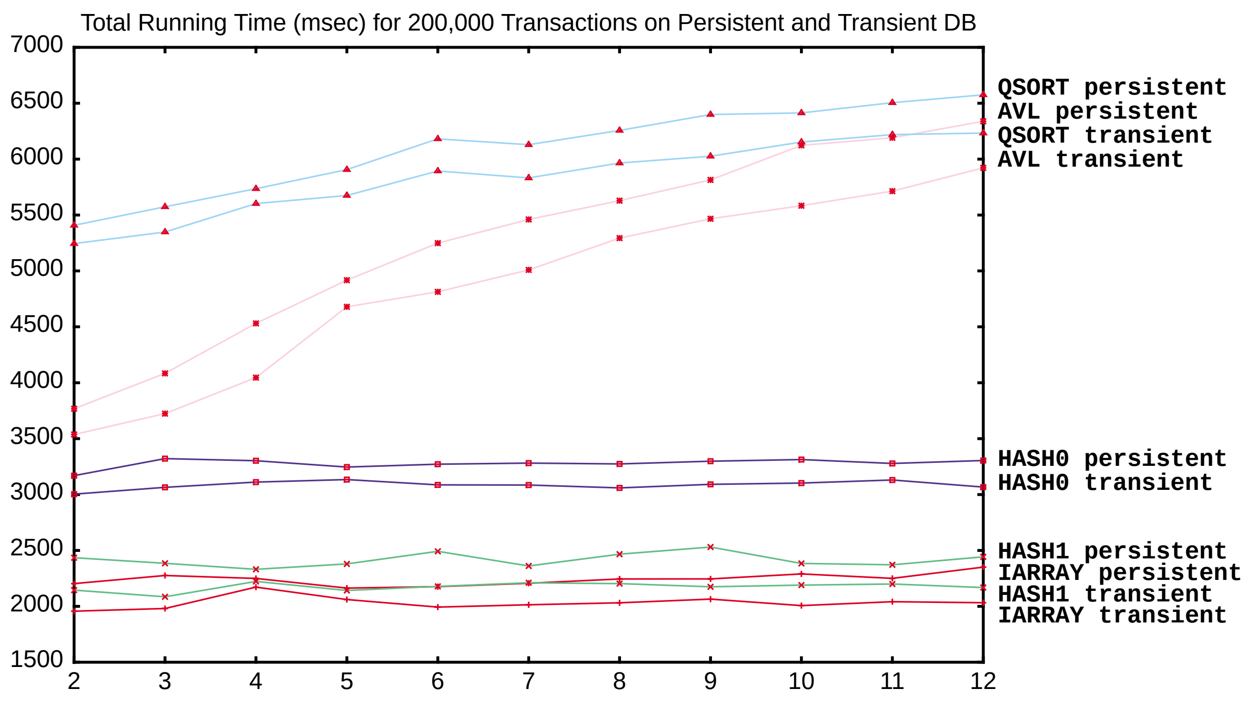


Figure 2: Total running time (msec) and speedup (relative to **`SMALLBASE`**) versus scale factor (relation size) for 200,000 transactions on transient databases.



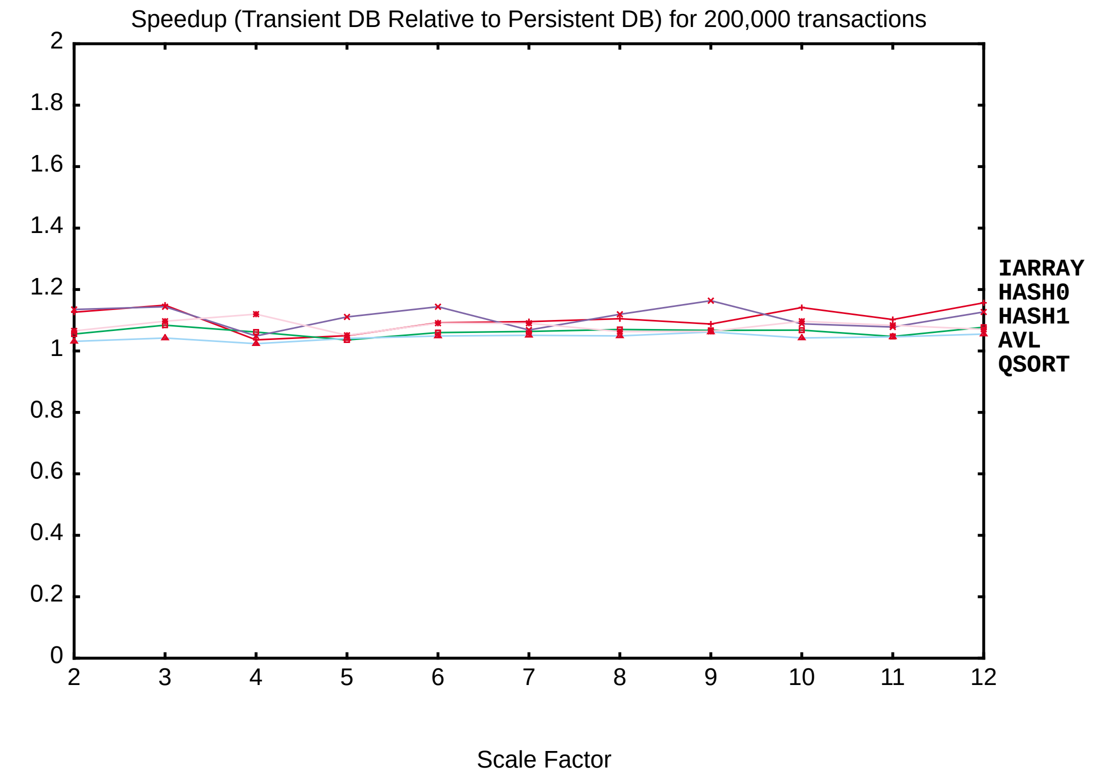


Figure 3: Total running time (msec) and speedup (persistent relative to transient) versus scale factor (relation size) for 200,000 transactions on P2.

5 References

[Bat94a] D. Batory, B. Geraci, and J. Thomas, “Introductory P2 System Manual”, Technical Report TR-94-26 , Department of Computer Sciences, University of Texas at Austin, November 1994.

[Bat94b] D. Batory, B. Geraci, and J. Thomas, “Advanced P2 System Manual”, Technical Report TR-94-27 , Department of Computer Sciences, University of Texas at Austin, November 1994.

[Bat95] D. Batory and J. Thomas, “P2: A Lightweight DBMS Generator”, Technical Report TR-95-26, Department of Computer Sciences, University of Texas at Austin, June 1995.

[Hew95] “Smallbase API Reference Manual (Smallbase 4.1)”, Database Technology Department, Hewlett-Packard Laboratories, Palo Alto, April 1995.

[Hey94] M. Heytens, S. Listgarten, M. Neimat, K. Wilkinson, “Smallbase: A Main-Memory DBMS for High-Performance Applications (Release 3.7)”, Database Technology Department, Hewlett-Packard Laboratories, Palo Alto, December 1994.

[TPC92] Transaction Processing Performance Council (TPC), “TPC Benchmark B: Standard Specification (Revision 1.1)”, March 1992.