SoftBase
Division of Fresche Solutions

z/OS Db2 Batch Design for High Performance

PROVEN

Technology

Tools Partner

Introduction

Neal Lozins

SoftBase Product Manager

All tests in this presentation were run on a dedicated zBC12 server

We used our products, Db2 DeadLock Advisor, Db2 Batch Analyzer, and TestBase to develop and monitor the tests shown here

Who is SoftBase?

DB2 z/OS testing and batch processing solutions

One of the original independent providers

- Founded 1987 in Asheville, NC

Who uses SoftBase?

- 5 of the top 10 US banks
- Many Federal and State Government Agencies
- 30% of private sector SoftBase customers are Fortune 500

Why Batch vs On-line Transactions?

Most Companies Use Batch Processing to:

- Perform sporadic maintenance
- Perform Calendar related tasks
- Use night time cycles
- Garner better performance and throughput

Most batch processes can be done online with random access.

Some examples:

- Statement generation
- Billing
- Cash posting
- Claims
- . . .

The major reason to use batch is:
Performance and Throughput

Why Tune Batch?

- Batch window gets smaller and smaller
- Risks of going outside the batch window
- IBM peak usage charge algorithm
- Many batch jobs are mission critical – billing, cash posting, …
- Batch often uses more resources than anything else in the shop

I/O Bound vs CPU Bound

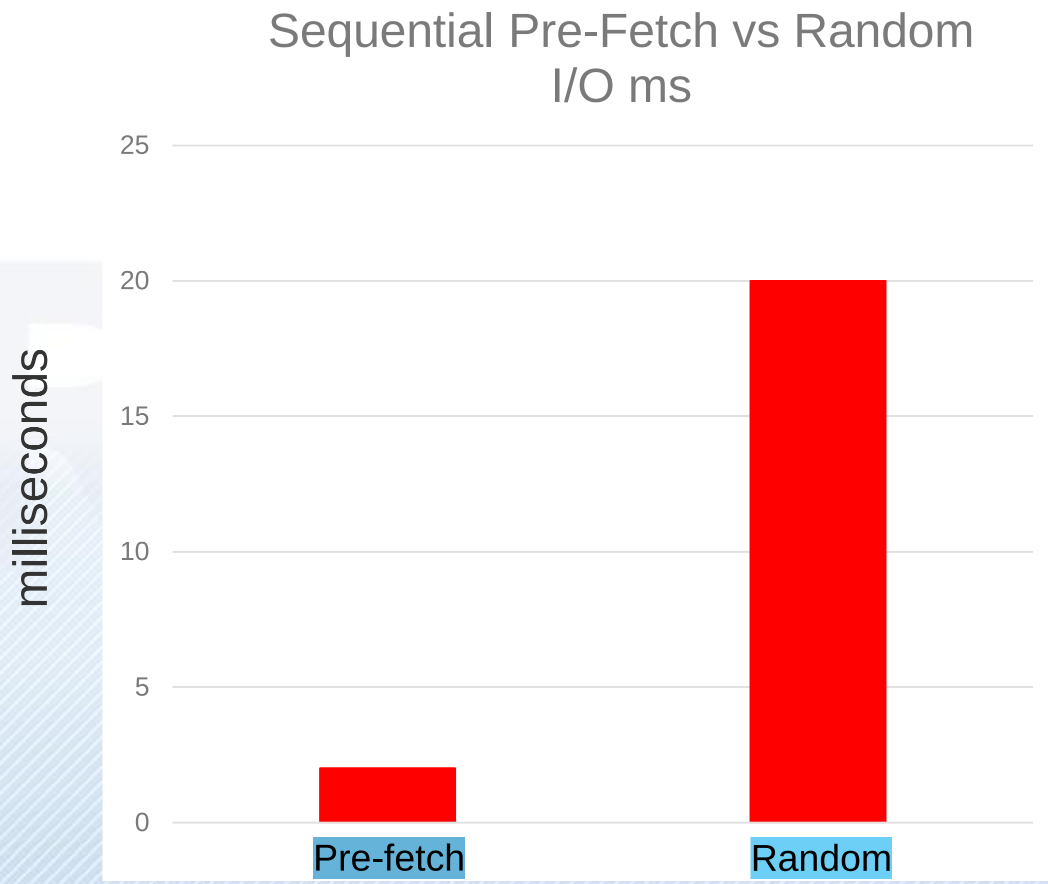
I/O Bound: A process is said to be I/O bound when trying to get more done causes additional or slower I/O. An I/O bound process can benefit from faster I/O subsystems but not by adding CPU power.

CPU Bound: A process is said to be CPU bound when no additional throughput can be garnered without adding CPU resources. A CPU bound process can benefit from adding CPU resources but not by faster I/O subsystems.

Random transactions (transactions in random order) are generally I/O bound while transactions done in order by the clustering key are generally CPU bound.

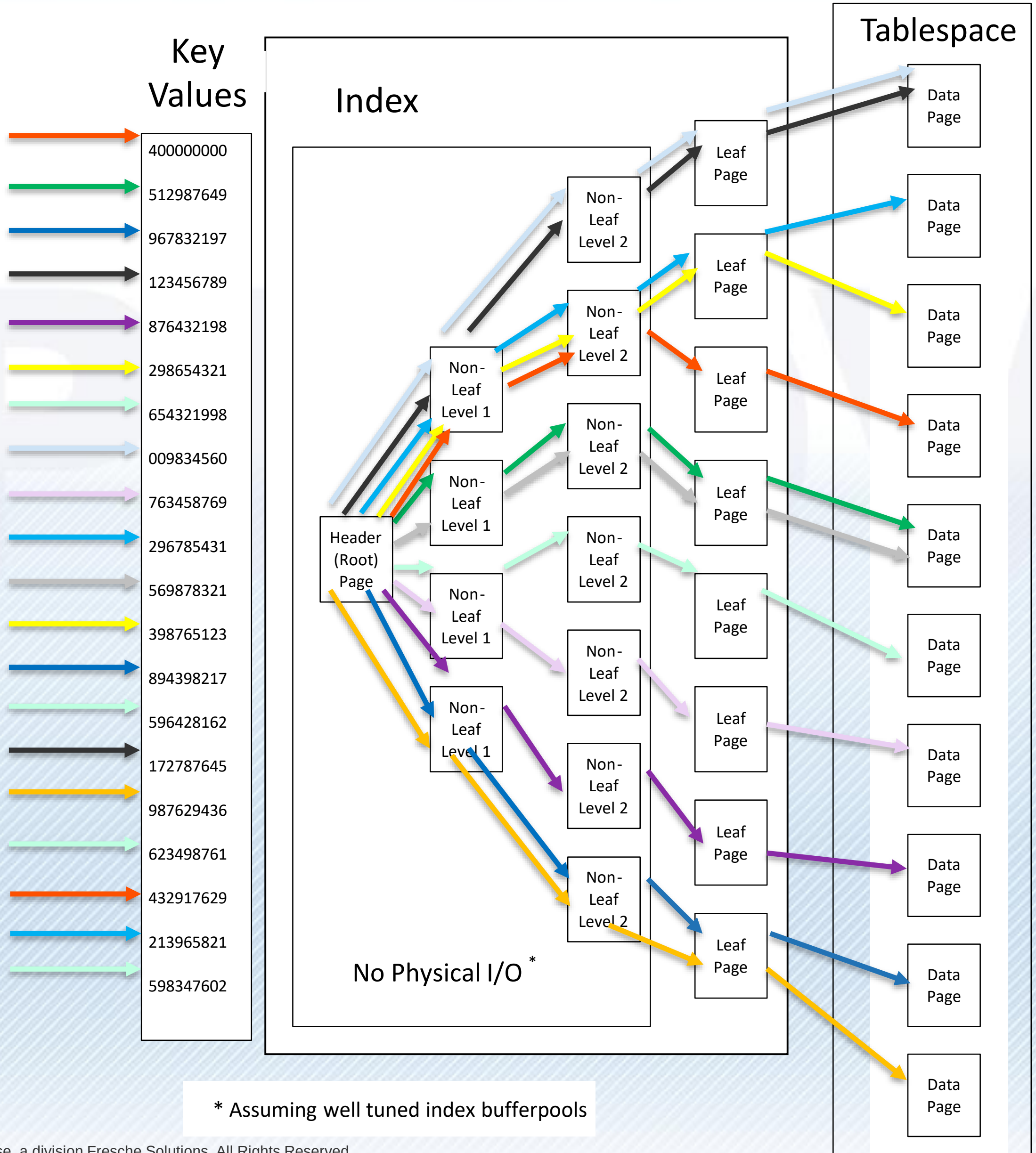
As we will see, a good batch design is typically CPU bound. It's much easier to add CPU cycles than it is to tune the physical limits of an I/O bound subsystem.

Pre-Fetch vs Random I/O



Pre-fetch usually averages 2ms per page. Random access can be tuned to be less than 20ms but usually rises with high activity against the same dataset. For very high activity it can be 100ms or more. Partitioning can be used to move the I/O to several datasets instead of a single dataset.

Random I/O

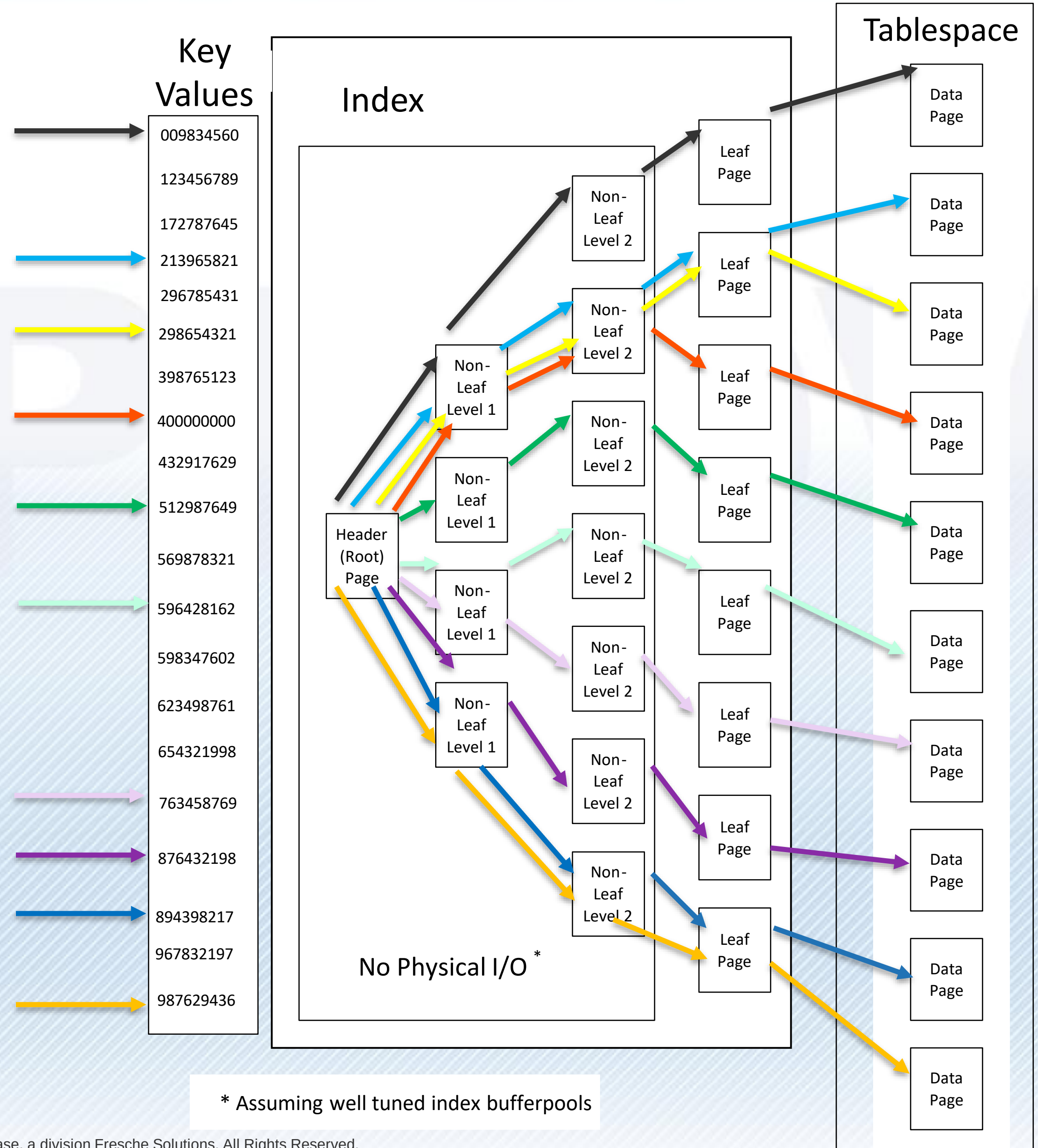


* Assuming well tuned index bufferpools

40 Physical I/O's:
  20 to the index leaf pages
  20 to the tablespace pages

Might get some accidental hits in the bufferpool

Sequential Pre-Fetch I/O



18 Pre-Fetch I/O's
8 Index Leaf Pages
10 Tablespace Pages

Not only do they take 1/10 the time, there are fewer I/Os !!!!

Types of Pre-fetch

- Sequential Pre-fetch – S in the plan_table
- List Pre-fetch – L in the plan_table
- Dynamic Sequential Pre-fetch – determined at run time
- Skip Sequential Pre-fetch – determined at run time

Sequential pre-fetch happens for large result sets ordered by the clustering index or simple tablespace scans.

List pre-fetch is to read the index to satisfy query results and mostly not a concern in this discussion. The other types of pre-fetch are very useful for well designed batch applications.

Dynamic and Skip sequential pre-fetch happen when processes are done in the order of the clustering index and do not require large result sets, but rather predictable access in the order of the clustering index.

What this means for good batch design

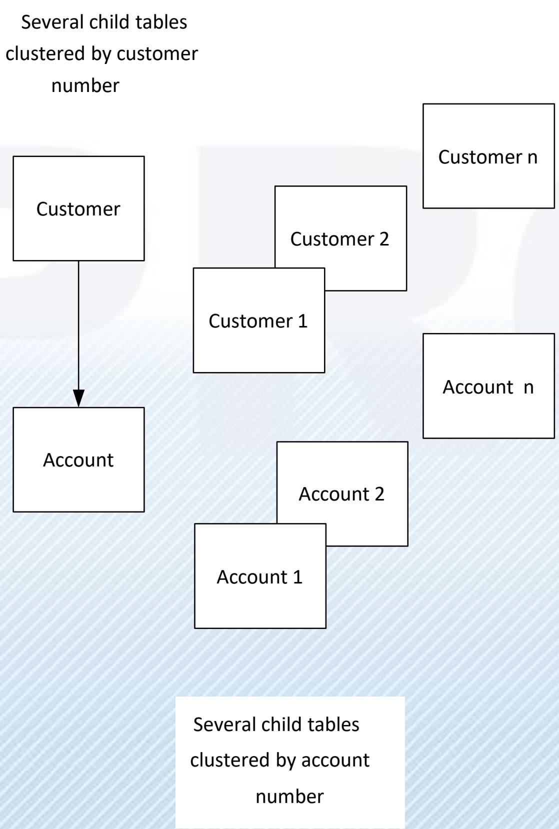
- Use sequential pre-fetch to get an entire set of rows
- When sequential pre-fetch is not available – order transactions by the clustering index to get dynamic or skip sequential pre-fetch
- If there are multiple clustering keys involved, split the process into as many sub processes as necessary and extract / sort by each process's clustering key
- Plan for purge or archive ahead of time – delete before insert or drop of partitions

As always:

- Verify access paths – to be the clustering index
- Checkpoint / Restart considerations
- Retry SQLCODE -911

Banking Data Model

Banking Data Model



We will see that data accumulated from the account processing and posted to the customer is better done by writing the key and the data to a flat file and sorting by customer number before posting the data to the customer table. An example might be the customer balance.

The banking data model is fairly straight forward. Customers and accounts for the most part. Complexity is in the number and types of accounts – DDA demand deposit (checking), savings, xmas club, brokerage, auto loans, personal loans, mortgages, etc.

Sample DDL and Program

```
CREATE TABLE VOLUME0.CUSTOMER
   (CUST_N       CHAR(10)  NOT NULL
   ,CUST_TYPE_N SMALLINT NOT NULL
   ,CUST_ADDR1   VARCHAR(30)   NOT NULL
   ,CUST_CITY    CHAR(20)      NOT NULL
   ,CUST_STATE   CHAR(02)      NOT NULL
   ,CUST_ZIP     CHAR(10)      NOT NULL
   ,CUST_PHONE   DECIMAL(10)   NOT NULL
   ,CUST_NAME    VARCHAR(30)   NOT NULL
   ,CUST_STUFF   VARCHAR(100)  NOT NULL
   ,CUST_START_DATE  DATE      NOT NULL
   ,CUST_BAL     DECIMAL(15,2)
   ,PRIMARY KEY (CUST_N)
   )
   IN CUST0.VCUSTS00
 ;
  CREATE UNIQUE INDEX
  VOLUME0.CUST01CU
             ON VOLUME0.CUSTOMER
            (CUST_N  )
  USING STOGROUP SBSIX01
  PRIQTY 400
  SECQTY 40
  ERASE NO          FREEPAGE 0
  PCTFREE 10
  CLUSTER
  BUFFERPOOL BP0
  CLOSE NO
      ;
```

Sample Program reads from transaction file and updates the balance column of the customer

```
210000-UPDATE.
    MOVE F-INPUT-CUST-NO  TO W-CUST-NO
    MOVE F-INPUT-CUST-BAL TO W-CUST-BAL.
    EXEC SQL
      UPDATE
      VOLUME0.CUSTOMER
      SET CUST_BAL = CUST_BAL + :W-CUST-BAL
      WHERE CUST_N = :W-CUST-NO
    END-EXEC.
```

This program was run several times with and without the transaction file sorted by CUST_N

Sample Program - Results

	Random		Pre-fetch (Sorted)		Improvement	
Number of Transactions	CPU Time	Elapsed Time	CPU Time	Elapsed Time	CPU Time	Elapsed Time
10,000,000	33.50	114.50	11.50	13.5	65.7%	88.2%
1,000,000	3.50	11.70	1.05	1.45	70.0%	87.6%
100,000	0.35	1.20	0.15	0.33	56.0%	72.5%
10,000	0.038	0.14	0.023	0.08	40.4%	42.9%

Run against 10,000,000 row table
Commit Frequency - every 2,000 updates in all cases
Times in minutes

```
 DB2 SQL Debug Explain Access Paths          Lines     1 of      3
Command ==>                                                Scroll ==> PAGE
                                                     DB2 Subsystem ==> DBCG

    S
   SE         T
   UL P       A                    LM            Line Commands:
   BE L  M S  B                    OO  SORT SORT   S - Select
    C A  IOE  N              J P F CD  NEW  COMP   X - Expand
    T N  XPQ  O  TT AT IX MC M F E KE  UJOG UJOG   / - List Commands
-- -- -- --- -- -- -- -- --- - - - --- ---- ----
   01 01 000 01 T  I  N  01  0      IX NNNN NNNN
__  TABLE: CUSTOMER
__  INDEX: CUST01CU
```

Conclusion:

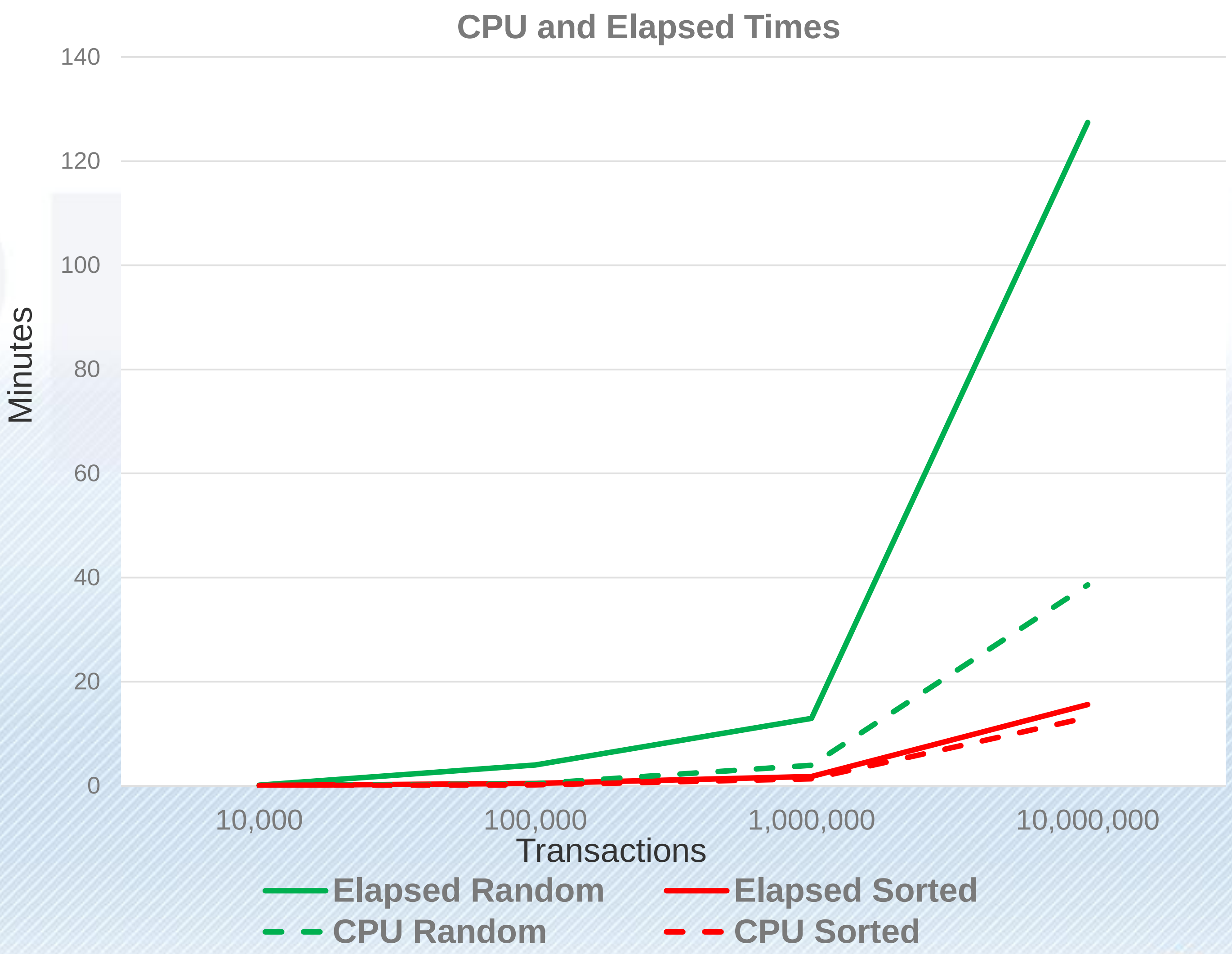
Sorting the transaction file is almost always worth the effort. It may invoke Dynamic Sequential Pre-Fetch or Skip Sequential Pre-Fetch - even when the Pre-Fetch column in the plan_table is blank.

The Random case is I/O bound because CPU time and elapsed times are far apart.

The Pre-Fetch case is CPU bound because CPU time and elapsed time are close.

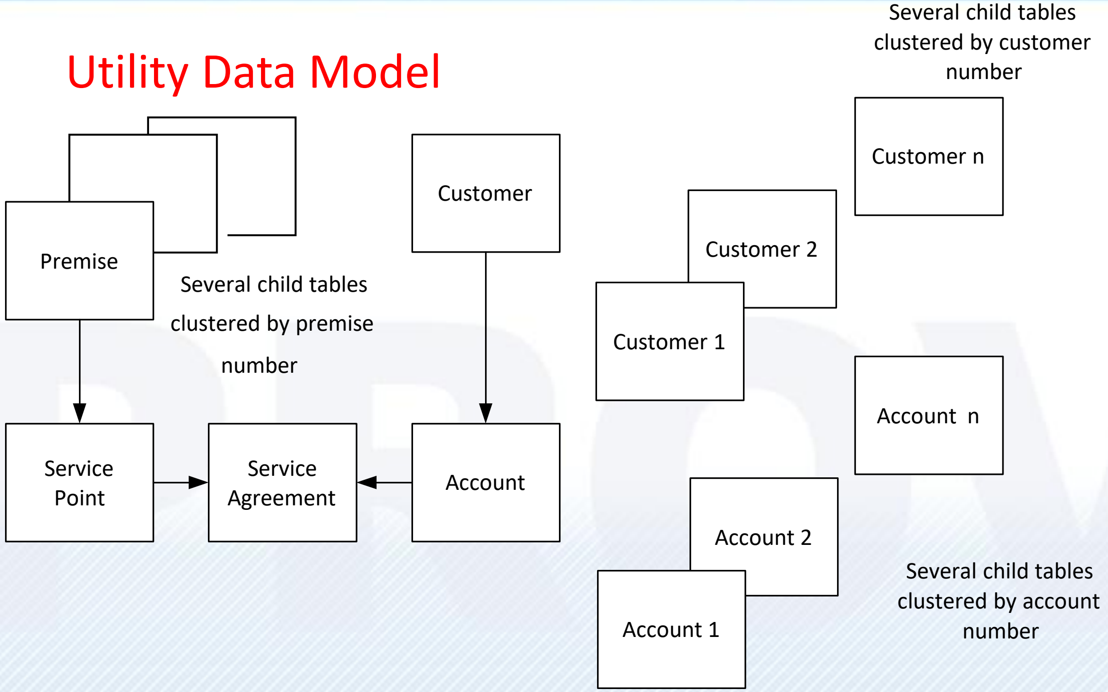
SoftBase
Proven Technology. Proven Tools. Proven Partner.

CPU and Elapsed Times



SoftBase
Proven Technology. Proven Tools. Proven Partner.

Utility Data Model



Getting the premise related data in the middle of billing makes the whole billing process I/O bound. Extracting the premise related data needed in premise order and sorting by account number before billing is started solves that. The data can be loaded into a table or kept in a flat file. The important part is that it be sorted by account number for billing purposes.

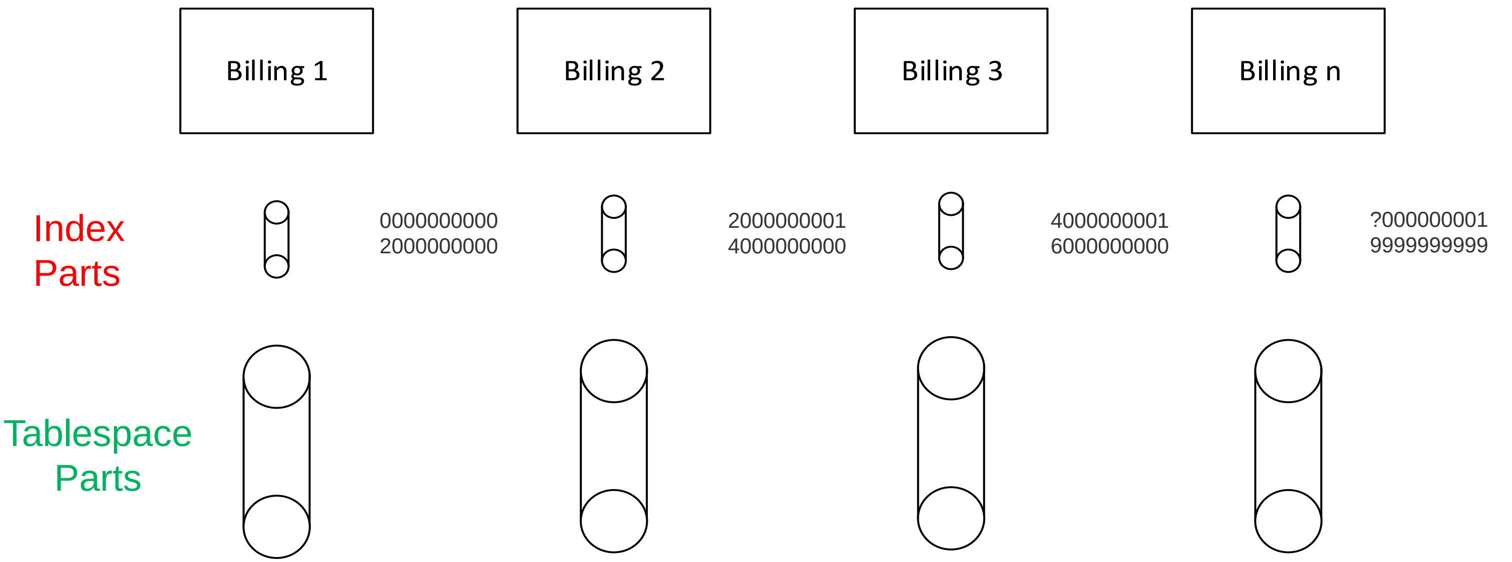
The utility company data model is a little more complex. It introduces a premise and a many to many relationship between premises and accounts. In order to bill an account, premise related data is required – especially meter readings at the service point, but a lot of other data as well.

Key Assignment algorithms for Clustering

- Do Cascade clustering keys to child tables – all account related tables start with account number
- Consider assigning keys so they cluster properly – include cycle as part of the key – can limit flexibility and can impact good business practices. Still need to track the premise across all the meter reading cycles etc.
- Consider assigning keys such that they are part of other keys. (eg - An account number that has premise in it.) Can prevent good business practices as well. Multiple premises billed on the same account. Service points billed on different accounts.
- Need historical record of the changes in keys that must be processed asynchronously (random I/O).

Consider / Use Parallel Processing

Today's z/OS mainframes have multiple CPU's, each of which is capable of servicing one and only one TCB (batch job) at a time. To garner even more throughput, we can take a CPU bound process and divide it into key ranges for processing in multiple batch processes so multiple engines can work at the same time. These key ranges can also be used as partition ranges. This has the advantage of easing possible lock contention and making utility processing and the application processing look at the same data. REORG and Image Copy can be scheduled as part of the application process by partition.



The choice of 'n' depends largely on the number of CPU's. Each billing instance could process multiple parts so more CPU's could be added at a later date.

I/O Bound Parallel Processing

Why not use parallel processing against I/O bound processes instead of CPU bound processes?

- Prone to deadlocks -911 especially if the synchronous I/O (random I/O) is for updates
- Might require row level locking to resolve deadlocks
- Still may get -911 with duplicates allowed indexes or updates across multiple clustering keys
- Can still be a solution if properly designed but starts to look like CICS transactions rather than batch
- Bufferpool support becomes a challenge due to all the synchronous I/O (random I/O) – more random I/O against the same tables can cause that 20ms number to increase dramatically. Range partitioning can help.

Additional Performance Considerations

- Use shorter keys – integer or decimal instead of character keys – makes for fewer levels and smaller indexes as well as more rows per page

- Pad duplicates allowed indexes with primary key to make them unique and to avoid large rid chain updates

- Pass data in linkage rather than using SQL to retrieve it every time it is needed. For example, the customer and account data is needed in most billing programs – keep a copy in storage and pass it around rather than getting it over and over

- Combine processes that access the same data

- Use new SQL improvements – outer join, WITH expressions, multi-row MERGE, multi-row INSERT, multi-row FETCH

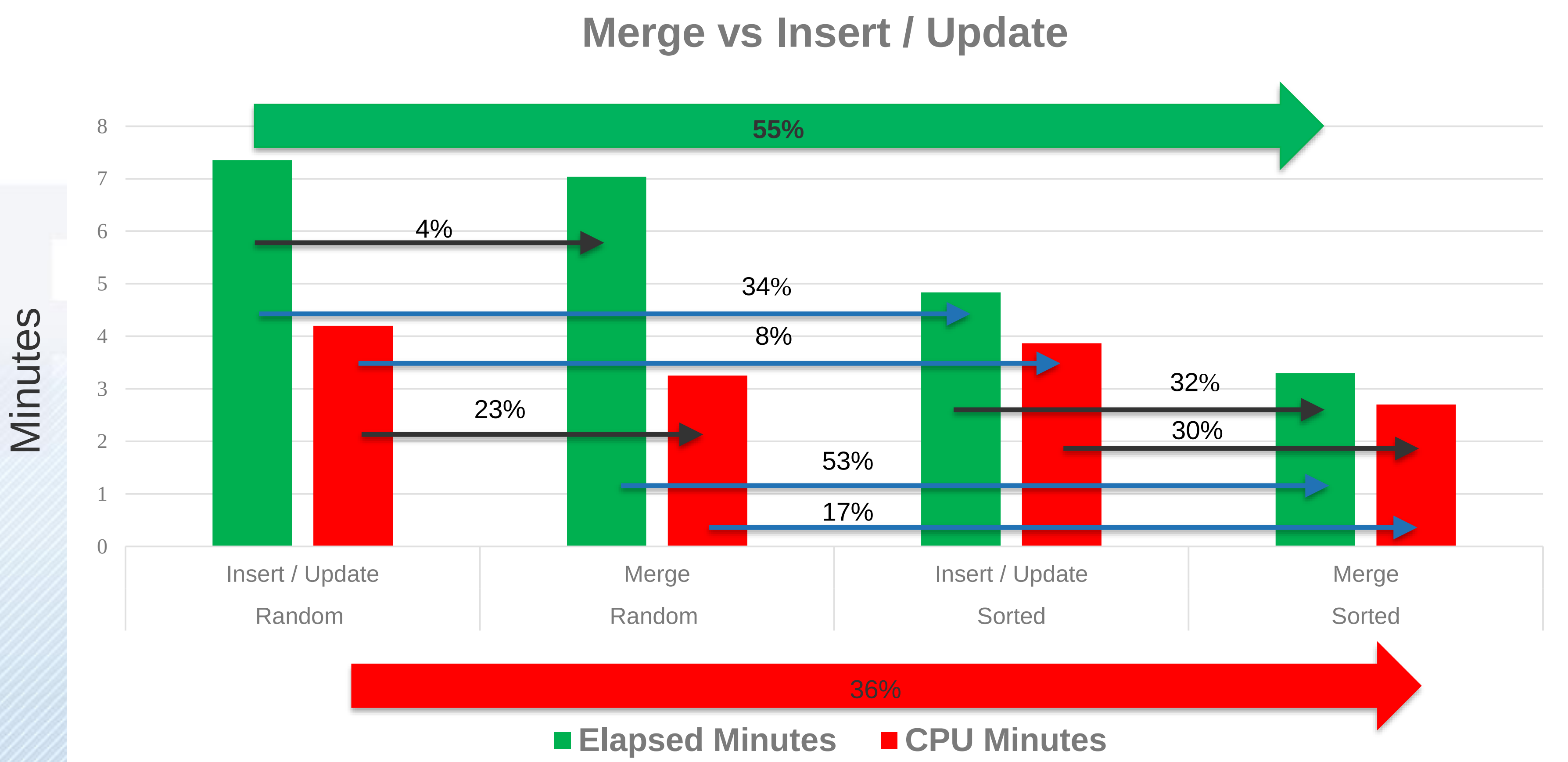
Remember:

1. Every SQL call avoided is potential CPU and I/O savings

2. Many of these improvements are against a single table or a single SQL – while the localized improvement can be huge, the overall improvement can be far less dramatic. Sometimes they can even make matters worse. Adding multi-row fetch to a well tuned cursor in an application that also has an I/O bound cursor can make it even more I/O bound.

SoftBase
Proven Technology. Proven Tools. Proven Partner.

MERGE Improvement over Insert / Update



Each of the 4 tests had:

500K Inserts
500K Updates
9M Result Table

SoftBase
Proven Technology. Proven Tools. Proven Partner.

Single Row Fetch vs Multi-Row Fetch

```
CREATE TABLE VOLUME0.CUSTOMER
   (CUST_N        CHAR(10)  NOT NULL
   ,CUST_TYPE_N SMALLINT NOT NULL
   ,CUST_ADDR1   VARCHAR(30)    NOT NULL
   ,CUST_CITY    CHAR(20)       NOT NULL
   ,CUST_STATE   CHAR(02)       NOT NULL
   ,CUST_ZIP     CHAR(10)       NOT NULL
   ,CUST_PHONE   DECIMAL(10)    NOT NULL
   ,CUST_NAME    VARCHAR(30)    NOT NULL
   ,CUST_STUFF   VARCHAR(100)   NOT NULL
   ,CUST_START_DATE   DATE      NOT NULL
   ,CUST_BAL     DECIMAL(15,2)
   ,PRIMARY KEY (CUST_N)
   )
   IN CUST0.VCUSTS00
CCSID          EBCDIC
;
CREATE UNIQUE INDEX VOLUME0.CUST01CU
 ON VOLUME0.CUSTOMER
  (CUST_N  )
  USING STOGROUP SBSIX01
    PRIQTY 400
    SECQTY 40
    ERASE NO
    FREEPAGE 0
    PCTFREE 10
```

Access to account via customer
number is still clustered
Because the key was cascaded

```
CREATE TABLE VOLUME0.ACCOUNT
     (CUST_N       CHAR(10)  NOT NULL
     ,ACCT_N       SMALLINT NOT NULL
     ,ACCT_ADDR1   VARCHAR(30)    NOT NULL
     ,ACCT_CITY    CHAR(20)       NOT NULL
     ,ACCT_STATE   CHAR(02)       NOT NULL
     ,ACCT_ZIP     CHAR(10)       NOT NULL
     ,ACCT_PHONE   DECIMAL(10)    NOT NULL
     ,ACCT_NAME    VARCHAR(30)    NOT NULL
     ,ACCT_NICKNAME VARCHAR(100) NOT NULL
     ,ACCT_NOTES     VARCHAR(300) NOT NULL
     ,PRIMARY KEY (CUST_N,ACCT_N)
     ,FOREIGN KEY FK1 (CUST_N) REFERENCES
      VOLUME0.CUSTOMER ON DELETE RESTRICT
     )
      IN CUST0.VACCT00
     CCSID          EBCDIC
     ;
      CREATE UNIQUE INDEX VOLUME0.ACCT01CU
      ON VOLUME0.ACCOUNT
         (CUST_N                     ASC
         ,ACCT_N ASC
         )
       USING STOGROUP SBSIX01
        ERASE NO
        FREEPAGE 0
        PCTFREE 10
        CLUSTER
        BUFFERPOOL BP0
        CLOSE NO
        PIECESIZE 2G
;
```

```
CREATE TABLE VOLUME0.ACCOUNT_1
     (ACCT_N       CHAR(10)  NOT NULL
     ,CUST_N       CHAR(10)  NOT NULL
     ,ACCT_ADDR1   VARCHAR(30)    NOT NULL
     ,ACCT_CITY    CHAR(20)       NOT NULL
     ,ACCT_STATE   CHAR(02)        NOT NULL
     ,ACCT_ZIP     CHAR(10)       NOT NULL
     ,ACCT_PHONE   DECIMAL(10)    NOT NULL
     ,ACCT_NAME    VARCHAR(30)    NOT NULL
     ,ACCT_NICKNAME VARCHAR(100) NOT NULL
     ,ACCT_NOTES     VARCHAR(300) NOT NULL
     ,PRIMARY KEY (ACCT_N)
     ,FOREIGN KEY FK2 (CUST_N) REFERENCES
      VOLUME0.CUSTOMER ON DELETE RESTRICT
     )
      IN CUST0.VACCT01
     CCSID          EBCDIC
     ;
CREATE UNIQUE INDEX VOLUME0.ACCT1ACU
ON VOLUME0.ACCOUNT_1
   (ACCT_N                ASC
   )
USING STOGROUP SBSIX01
  ERASE NO
  FREEPAGE 0
  PCTFREE 10
  CLUSTER
  BUFFERPOOL BP0
  CLOSE NO
  PIECESIZE 2G
 ;
CREATE INDEX VOLUME0.ACCT1BNU
ON VOLUME0.ACCOUNT_1
 (CUST_N                 ASC
 ,ACCT_N                 ASC
 )
USING STOGROUP SBSIX01
  ERASE NO
  FREEPAGE 0
  PCTFREE 10
  BUFFERPOOL BP0
  CLOSE NO
  PIECESIZE 2G                            ;
```

Access to account via customer number is NOT clustered

Single Row Fetch vs Multi-Row Fetch

Program Opens a cursor on first 500,000 customers and for each row fetched – fetches the corresponding account rows which on average are 5 for a total of 2.5 million account rows.

Obviously, we could make this an outer join and get some performance benefit, but here we are attempting to show the improvement from multi-row fetch.

	MRF array	CPU	Elapsed	CPU Improvement	Elapsed Improvement
Clustered	0 – no mrf	2.73	3.5		
	100	2.46	3.2	9.9%	8.6%
Random	0 – no mrf	5.88	10.3		
	100	5.56	9.9	5.4%	3.9%

In this case, we got some improvement from MRF even in the random test. Had the random access been even worse, we might have even seen a worsening with MRF because it drives the I/O bound process even harder making it even more I/O bound.

Increasing the buffer size to 1,000 yielded substantially similar results as 100.

Nice to Do – all the additional performance considerations

- **Use shorter keys** – integer or decimal instead of character keys – makes for fewer levels and smaller indexes as well as more rows per page
- **Pad duplicates allowed indexes** with primary key to make them unique and to avoid large rid chain updates
- **Pass data in linkage** rather than using SQL to retrieve it every time it is needed. For example, the customer and account data is needed in most billing programs – keep a copy in storage and pass it around rather than getting it over and over
- **Combine processes** that access the same data
- **Use new SQL improvements** – outer join, WITH expressions, MERGE, multi-row INSERT, multi-row FETCH, paging for multi-column keys WHERE (AC,EX,LN) > (:AC,:EX,:LN)

Remember:

1. Every SQL call avoided is potential CPU and I/O savings
2. Many of these improvements are against a single table or a single SQL – while the localized improvement can be huge, the overall improvement can be far less dramatic

SoftBase
Proven Technology. Proven Tools. Proven Partner.

Must do's

1. Process in clustering index order

2. Extract and sort needed data in clustering index order to remove random I/O

3. Use Parallel Processing to get multiple CPU's involved in the process

Implementing the first 2 can turn an I/O bound process into a CPU bound process that can run 10 times faster or more. The third, offers improvement factors up to the number of CPU's available – typically 5 times or more.

Thank you!

Thanks for you time today!

Questions?

www.softbase.com

800-669-7076