Experiences Using the JCC Logminer Loader

Oracle Rdb
Technical Forums

Jeffrey S. Jalbert
Thomas H. Musson,
JCC Consulting, Inc.

Abstract

The LogMiner capabilities recently added to Rdb provide a platform to build uniquely new architectures for managing and deploying Rdb databases. Applying this capability requires a new tool to properly utilize these capabilities.

The JCC LogMiner Loader is this tool.

This presentation will focus on the Loader and share our experience in using this product to completely reorganize a significant (50 GB) production database (not counting snapshots) while requiring minimal down time.

We will also address the performance of the Loader in maintaining an Oracle 9i database.

Introduction – Loader Characteristics

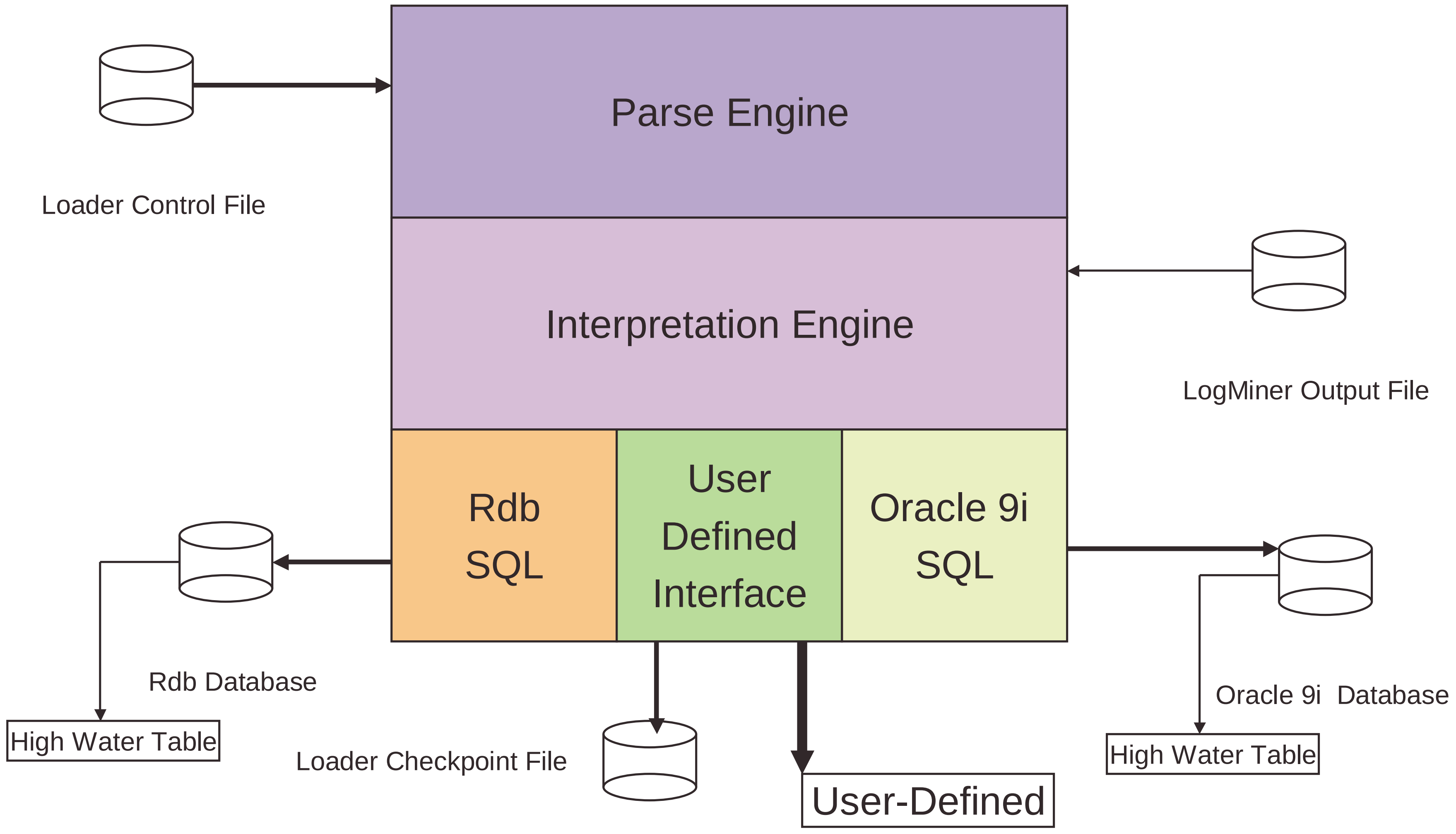
We wanted to accomplish three goals:

- Reorganize databases with minimal disruption to production
- Replicate changes from a production database to a target database without impacting the production database. The target database could be:
  - Rdb database
  - Oracle database
- Provide an API into callable routines that can transmit Rdb data into user-specific external data stores in a standard format

Loader Characteristics

- The JCC LogMiner Loader is a general-purpose tool to process Rdb LogMiner output files
  - Reads a Control file to determine what to do
  - Generates dynamic SQL to maintain either DSRI databases or OCI databases
  - Provides support for a user-defined transport
    - Generates XML formatted messages to pass to the interface, one message per transaction.
  - Maintains target data store in a transactionally consistent fashion

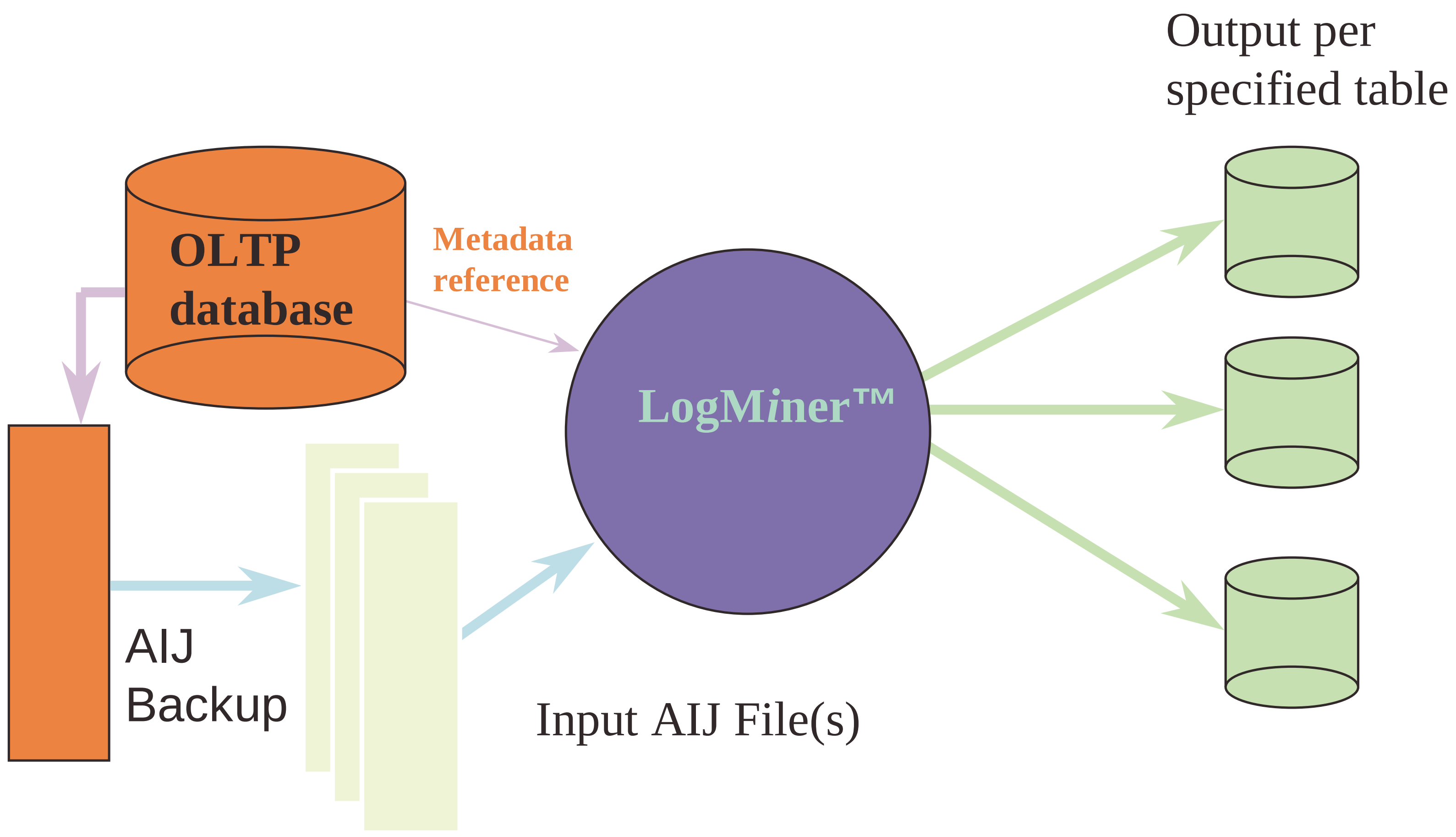
Loader Architecture



The Loader Control File

- The control file specifies the actions of the loader
  - Name of target
  - Tables and columns to be sent to target
  - Type of transmission [Rdb, Oracle, User-Defined]
  - Loader stream name
    - It is possible to run multiple copies of the Loader with the same target
    - Allows target db to be maintained from multiple source databases
    - Allows processing of target to proceed faster by splitting the input LogMiner file into multiple output streams
- The control file is a surrogate for reading the source database metadata directly
  - That database may not be accessible to the loader

Preparing the LogMiner Output File



Contents of the LogMiner Output File

- Contents
  - Data from user tables
  - Inserts and modifies
    - ‘Final’ record content per transaction
  - Deletes
    - Record content just prior to delete
- Committed transactions only
  - Rolled-back transactions ignored
- Final form of row returned
  - Transaction that inserts and deletes a row results in delete record being extracted
  - Row modified many times in one transaction results in a single modify output record

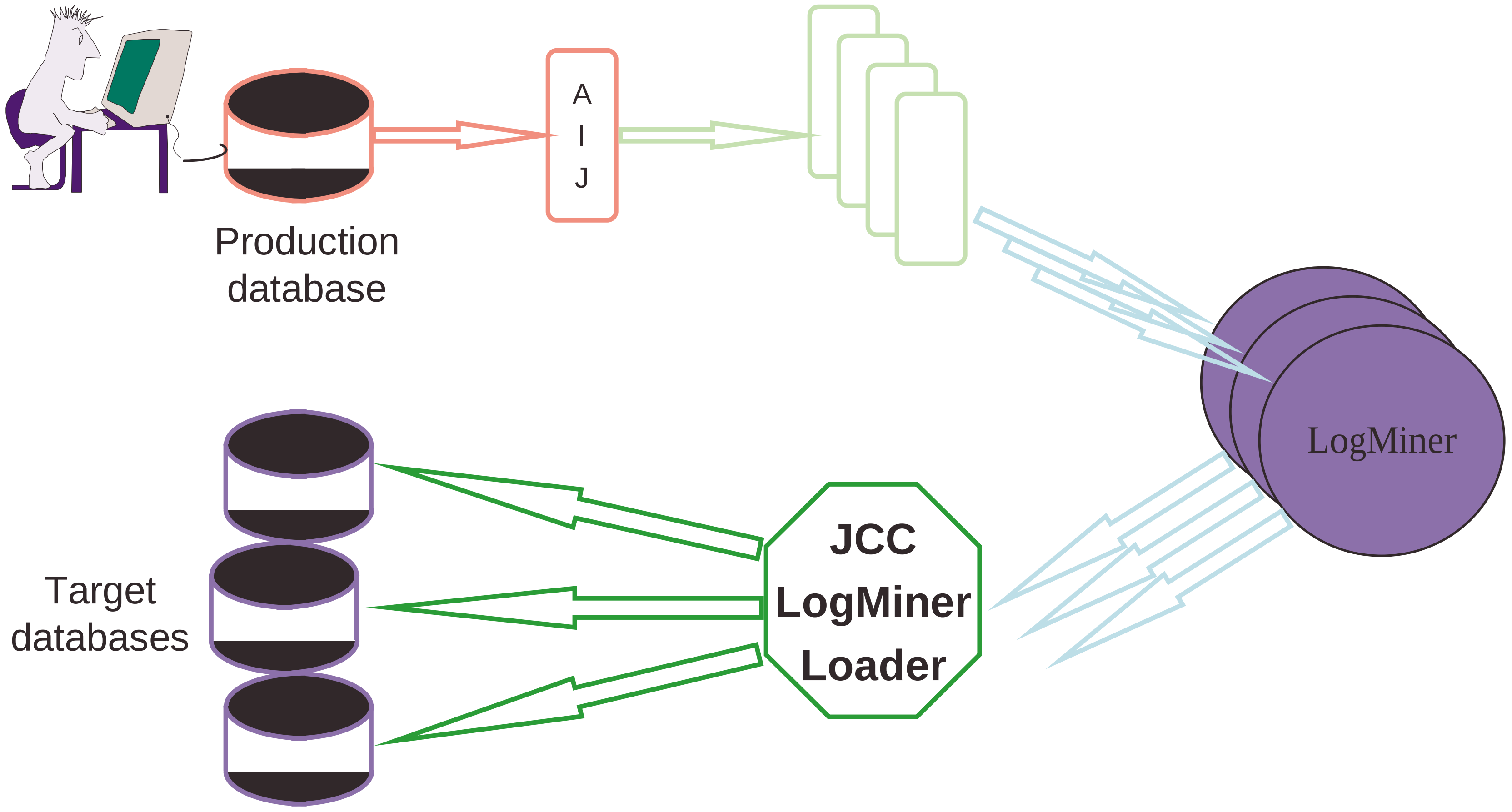
LogMiner Restrictions

- Blobs (segmented strings) are not supported
- Metadata changes are not allowed within an AIJ
- No quiet point AIJ backups are supported:
  - Can process a sequence of AIJs that start just after and end with a quiet point
- COMPUTED BY columns are not stored in the AIJ
- DROP or TRUNCATE do not include details in the AIJ
- Optimized AIJs, the logminer works on transaction boundaries
- Vertical record partitioning (VRP) is not supported

JCC LogMiner Loader

- Reads output from LogMiner for Rdb
  - All requested tables to the same output stream
- Applies to target database in transactionally consistent manner
- Transaction based commit interval specified in control file
  - Commit intervals wrap multiple source database transactions into a single target database transaction
  - Commit intervals greater than 1 allow increased performance in target database
- Uses SQL interface to target database
  - Lets Rdb/Oracle maintain the necessary structures (indices, SPAM, ABM)
- Uses XML formatted messages for user-defined interface
- Maintains high-water mark information to enable restart from failure
- Retries entry to the target, in case of deadlock
  - Buffers input records since last checkpoint

Where Does It Fit?



JCC LogMiner Loader Methodology

- Identifying the record structure
- Uniquely identifying records
  - Primary key
  - Unique index
  - Original DB key
- Dynamic SQL
- Target database structure
  - Constraints
  - Triggers
  - Indices

Identifying The Record Structure

- Record output from LogMiner for Rdb is variable length
  - 74 bytes fixed LM information
  - Remaining is variable length depending on the table
  - Very wide tables may require multiple records, RMS record size limitation
- Control file specifies the column data type and record layout for each table
  - Must reflect organization of “this” database
  - Allows wide control of the actions that the Loader performs
  - Metadata definition for the control file is tedious
    - DCL/SQL script generates initialization file automatically
      - Makes guesses at unique columns and primary keys
  - Supports references to other control files to be included in definition (“@” constructs)

Unique or Not?

- How does one update or delete a row in a database if there is no way to uniquely select that record?
- All tables loaded by the JCC LogMiner Loader must either:
  - – Have a unique column or set of columns
    - Usually called primary key
    - Keys cannot change because Loader does not have access to initial values of data
      - – Provide DB-key mechanism to get past this issue
      - – Ties target to physical organization of source db
        - Often is only transitory
  - – Be insert only

Work With Physical Structure

- Both the LogMiner and Loader work at a non-relational level of the database.
  - They have “guilty knowledge” of the database row structure.
  - Really working with database lines and not with SQL rows.
- Each Control file is closely associated with the source database organization

Uniquely Identifying Records

- Primary key
  - Only if application *does not modify* primary key columns
- Unique index
  - Only if indexed columns *can not change*
- Original DBKey
  - This is the DBKey of the row in the source database
  - Utilized and maintained by the JCC LogMiner Loader
  - Catch-all for those tables that do not have primary keys, unique indices, or that have columns within those that are updated
  - Assumes that Original DB Keys are not reused
    - Valid only for limited periods of time, usually

Original DBKey

- In cases where the table has
  - No primary key
  - No unique indices
  - *Not* insert-only
- We create a column in the target database table and populate it
- Could also be generated with RMU unload/RMU load

```
SQL> alter table add originating_dbkey char(8);

SQL> update table set originating_dbkey = dbkey;

SQL> create unique index table_t_01 on table
  (originating_dbkey) ...
```

SQL Interface

- Question: what is the most efficient way to write an SQL statement for Rdb such that an insert, update, or delete is completed?
- Possible answers:
  - Stored procedure, one per table
    - Static; Maintenance headache if tables change; Inflexible
  - Individual insert, update, deletes
    - Static; Maintenance headache if tables change; Inflexible
    - Dynamic; Three compiles; No maintenance
    - Multiple messages between Rdb and the Loader
  - Multi-statement procedure
    - Dynamic; Compiled only once; No maintenance

Dynamic Multi-statement SQL

- A single multi-statement procedure to insert, update, or delete a row in a table based on the LM Action field
- Handles ‘M’ being insert or modify by testing for existence
- Add original dbkey column reference if necessary

Order of Application of Changes

- If target database is to be slaved to source database
- If application code performs updates as:
  - Delete
  - Insert
- The data being read by the loader is in unpredictable order and the insert could occur before the delete
- If a unique index is placed on the target database table for, e.g. hash placement reasons, this will cause the loader to fail
- The loader stores all input for a transaction in memory and applies all delete operations prior to update operations

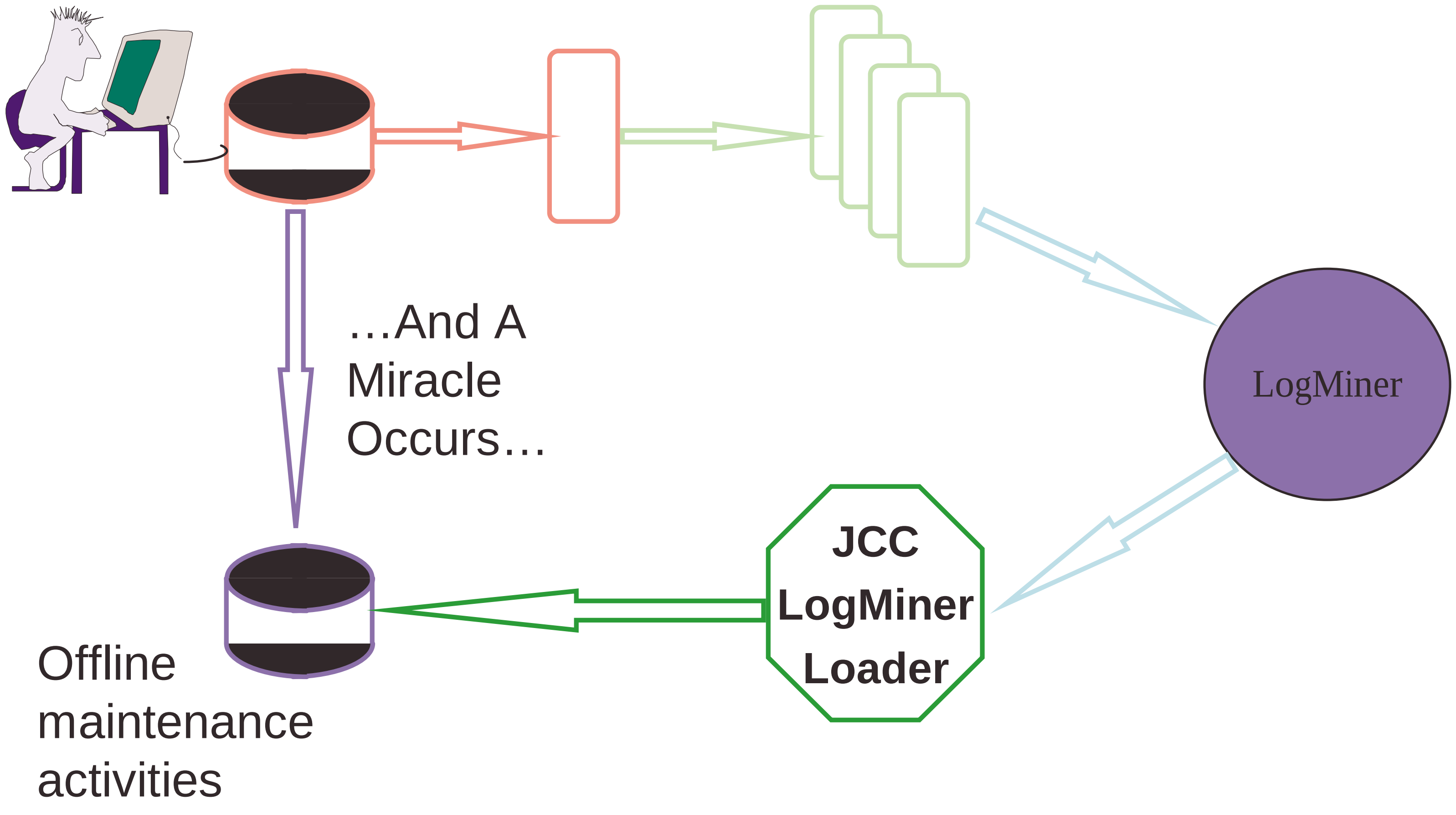
Target Database Structure

- Constraints
  - Can’t guarantee records in same order as original transaction
  - Must relax foreign key constraints and check constraints that perform a similar function (enforced on source database)
- Triggers
  - Effects of triggered actions are represented in AIJ
  - Drop triggers and replace prior to production release
- Indices
  - Unique columns must be honored
  - Anything you want
  - Indices to support original dbkey column (where needed)
- Column data types can be anything that is compatible

How To Use the Tools

- Must specify only a single output for LogMiner
  - Need to have all records in a transaction in the same input stream to maintain transactional consistency
  - For a complete rebuild individual transactions are not important. Can create parallel streams each with subsets of the tables.
- Target database can be local or remote
- Will automatically detect if last execution failed and will restart with the record after the last successful checkpoint

Minimal Downtime



The Eightfold Way of Database Restructuring

1. Restore a backup of the production database
2. Create target empty database with only hashed placement indexes. Add necessary DBKey columns if needed
3. Unload views containing DBKeys if necessary and load target database
4. AIJ backup(s) on production database, load data changes using LogMiner for Rdb and JCC LogMiner Loader in tandem
5. Repeat step 4 till downtime is reduced to acceptable value
6. Add remaining indexes
7. Repeat step 5
8. Add constraints and triggers

Other Applications of the Technology

- Maintaining a reporting database
  - Differing indexing
- Loading data into a Data Warehouse
  - Delete records can be transformed into updates
  - Updates and deletes can be transformed into inserts
- Maintaining multiple warm standby databases in remote locations
  - Same as the minimal downtime scenario
  - Could be Oracle 9i
- Maintain down level versions of database
  - Rdb 6.0 minimum for multi-statement SQL
  - Not regression tested
- Send data to a user-defined transport (e.g. a messaging system)

Current Status

- All testing to date has been with
  - Rdb V7.0-61(with latest LogMiner patches) as the target database
  - Oracle 9i (on Windows 2K) (using SQL*net transport)
  - User-defined transport is TCP/IP socket to another system
  - File input for the LogMiner output data
- Performance is much better than the application that wrote the data
  - No rolled back transactions
  - Only deal with data being changed (no extraneous selects)
  - No application logic to enforce
  - Restricted subset of tables for each stream possible
  - Large commit interval (more updates/commit)

Monitoring The Loader

- The Loader writes results of its actions into a global section
  - A real-time monitor is supplied to display progress
  - Can display summary or detailed activity
    - Number of rows & transactions processed
    - Progress of loader relative to source database
    - Sizes of rows transmitted (histogram format)
    - Duration of transactions in target (histogram format)
    - I/O & CPU costs
- RMU /SHOW Statistics
  - Repeated runs allow removal of Rdb physical design errors based on *actual* updates done to db

Futures for JCC LogMiner Loader

- Continuous LogMiner support
  - Maintain target databases in near real-time
- Multiple target databases concurrently
- Handle multiple record versions
- Access database directly for metadata description
  - If opened on machine loader is running on
- Suggestions?

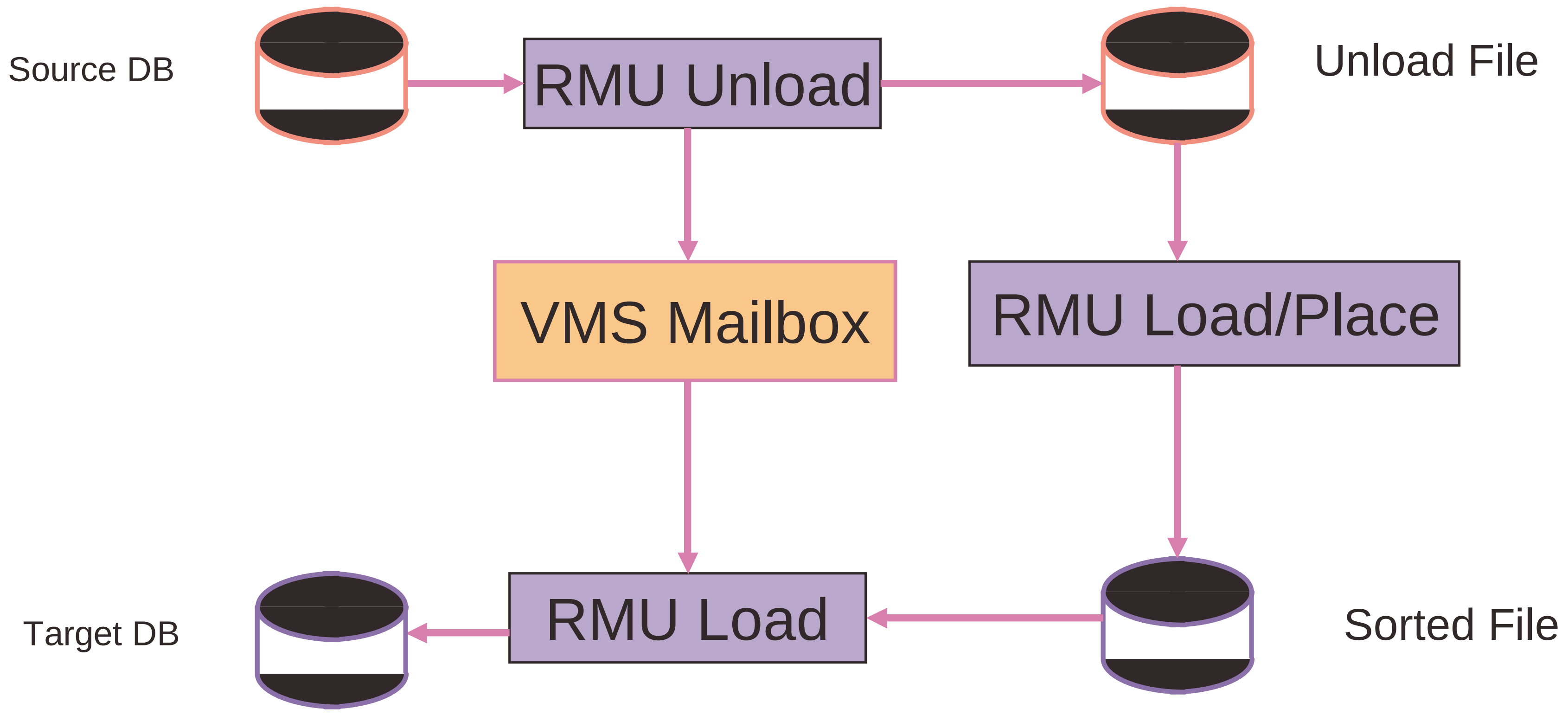
Case Study – Reorganize a Database

- 50 Gb database supporting a ½ million customer business
- 23 tables are placed in mixed format areas. Other 150 tables are in uniform areas
- Largest placed table is over 60 million rows, multiple 20-million row tables
- Number of customers has increased much faster than projected
  - RMU/MOVE area resulted in excessively large database areas and buffer sizes
    - Database buffer size is now 24 blocks.
    - Don’t feel that increasing that is reasonable anymore
    - Would prefer 12 block buffers for this TP application

New Target Database

- Sized for 1 million customers
  - Many tables are horizontally partitioned
  - One mixed area sized for full customer load
- Buffer size back to standard 12 blocks
- Rebuild is highly parallelized
  - Keep the CPUs busy
- For rebuild uses local buffers
  - All streams perform orthogonal work
- Will be switched back to system space global buffers for production

Unload Load Optimization



No Placement Placement Via

Case Study — Optimized Computing Environment

- Hardware is a dual-processor ES40 with 2 CPUs
- 5 Gb memory, all available for this activity
  - Create local memory disk for root file and RUJ writes during unload-load activity
- Locally attached fast wide SCSI disks on 6 controllers
  - Formed host-based RAID sets across controllers
- Reorganized database on two separate stripe sets
- Source for unload on separate stripe set
- Sort work files on 4 separate spindles

Sizes of AIJ and Recovery

- AIJ File size was 997,866 blocks long
- Unloaded a total of 649,014 modify records and 55,160 records
- Recover job indicated that it recovered 329,825 transactions
- LogMiner unload job ran for 7:19.20 and used 1:19.64
- JCC LogMiner Loader ran in
  - 13 parallel streams
    - Partitioned large tables into their own streams
  - 30 minutes for longest stream
  - Entire process was CPU bound

Parallel Rebuild

- Unload-load jobs run in 5 parallel streams
  - One table per stream
  - System becomes CPU bound
  - Required 7 ½ hours to perform unload-load on all tables with only hash placement indexes as necessary
- LogMiner Loader ran in 13 parallel streams
  - Used originating db-key methodology because application undocumented
  - Required addition of indexes on originating DBKey columns
    - Index build ran CPU bound in 14 parallel streams
    - Required 106 minutes to complete

Timing for Traditional Export/Import

Export Database	150 Minutes
Import Database	1,050 Minutes
Release to Production requires about 20 hours	

Timing for Traditional Unload-Reload

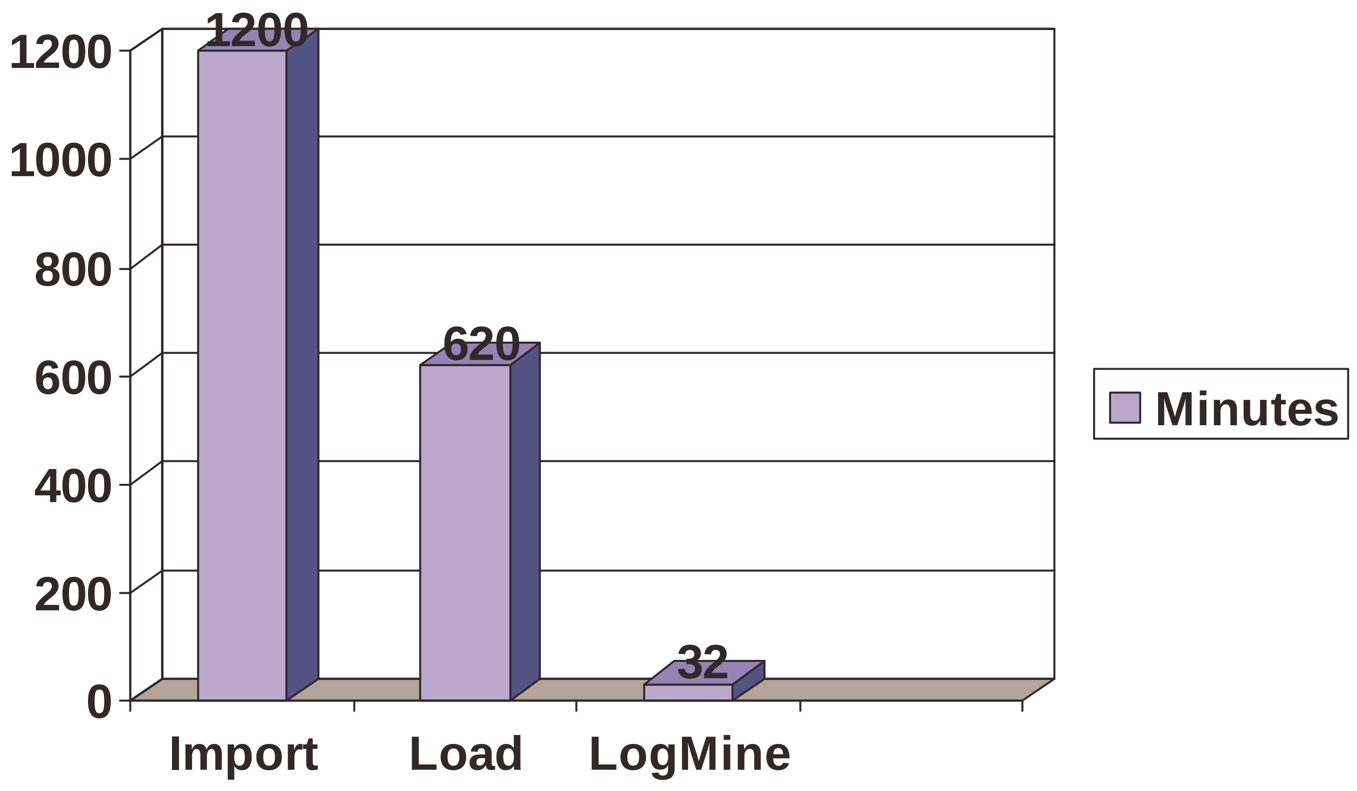
Import Empty database	25 Minutes
Restore Backup DB	120 Minutes
Unload-Load All Tables	440 Minutes
Add Normal Indexes	180 Minutes
Release to Production requires about 10 1/2 hours	

Timing for LogMiner Loader Approach

Import Empty database	25 Minutes
Restore backup DB	120 Minutes
Unload Load all tables	420 Minutes
Add DBKey Indices	106 Minutes
LogMine 1 day of AIJ	7 Minutes
Run JCC Loader	60 Minutes
Apply indexes	180 Minutes
Mine and rerun loader	30 Minutes
Triggers and constraints, DBKey indexes	2 Minutes

production is down

Downtime Comparison Between Methods



Verifying the Load

- Proof of correctness done by comparing all rows of all tables
  - Loaded database
  - Recovered database
- Use view which sorts table by all columns
- VMS differences of unload files (text format with special NULL string)
- Ran through 2 VMS mailboxes, one for each database
- Required about 20 wall-clock (and about 35 CPU) hours to verify all tables

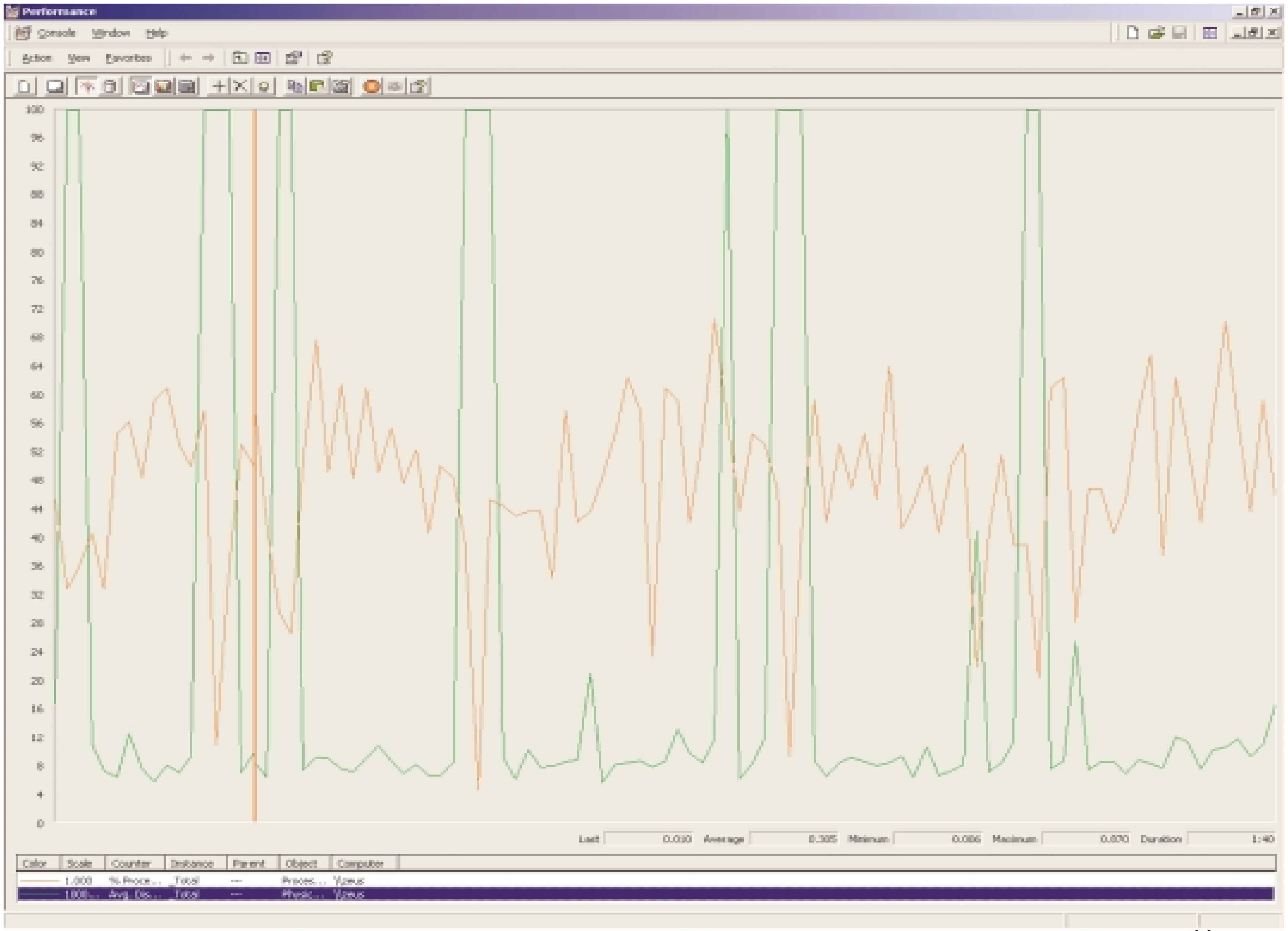
Oracle 9i Results

- Used same AIJ files as previous example
- Used same originating DB-key columns. For Oracle these were converted to integers instead of strings
- Used large commit intervals since this was regression testing
- 32-Tables being Loaded, used the full Loader output file
- Represented about 1/3 of the actual workload
- 100 Mhz network
- Target W2K machine was 400 Mhz processor, 512 Mb of memory

Performance of W2K Server

Disk

CPU



Last 0.010 Average 0.305 Minimum 0.006 Maximum 0.070 Duration 1:40

Color	Scale	Counter	Instance	Parent	Object	Computer
	1.000	% Proce...	_Total	---	Proces...	\\zeus
	1000...	Avg. Dis...	_Total	---	Physic...	\\zeus

Oracle 9i Results

- Single threaded updates to target database
- Updated 125 rows per second in target
- Used between 10% and 15% of 1 500 Mhz EV6 CPU on VMS side
- Performance limit was network bandwidth
  - Optimized by using large commit intervals
  - Would be optimized even more by specializing high-update tables to their own streams
    - Due to the way Oracle 9i handles prepared dynamic SQL statements
- Completed 24 hours of production system processing in 42 ¾ minutes.

User Interface Results

- We are at the earliest stages of testing the user-defined interface
- Using a very high volume update database
- Initial testing was to write to a file
  - Just ran CPU bound formatting the data
  - Ran 24 hours of processing in under 2 hours
- Initial testing indicates that we are using modest CPU resources and are limited by the performance of the user-provided interface.

Observations

- Reduced down time makes reorganization of database possible. Downtime well within interval for normal database maintenance
- Highly parallel operations allow system throughput to be maximized
- Elimination of file writes to disk improves system utilization
- Use of striped disks improves throughput of rebuild
- Use of memory disk for root file and small RUJ writes improves throughput
- Comparing all the data via VMS difference utility is very expensive.
  - We may write our own code to see whether something specialized may do better

Conclusions

- LogMiner for Rdb is a valuable tool that gives us access to the data stored in AIJ format
- JCC LogMiner Loader enhances the capabilities of the LogMiner
  - To support database reorganizations with minimal disruption and
  - Replicate changes to other (Rdb *or* Oracle) database(s) or
  - Replicate changes to other data stores
- Provides new dimensions for deploying applications

Field Test

- The JCC LogMiner Loader is in late field test
- We are soliciting a limited number of sites who will actively test this product
- Final production release is expected shortly
- Please direct inquiries to me at the E-mail address

Jeff@jcc.com

Rdb List Server

- Join the international discussion group about Rdb.
- Send E-mail to:

oraclerdb-request@jcc.com

- In the body of the message include the command:

Subscribe OracleRdb

Thanks Rdb Engineering!

The authors want to thank Rdb engineering for their close cooperation and support while we were trying the new LogMiner code and for adding features we all need…

Questions