HP Capstone
Big Data Analytics

Prepared by:
Nic Desilets
and
Nathaniel Whitlock

Introduction

The following slides will be an overview of our individual experiences between the end of fall term and week seven of winter 2017. Each of the two of us will read our individual sections and the rest will be split up.

Purpose and Goals

- Purpose
  - Research parameters and features relevant to analytical workloads.
  - Use this knowledge to fix main problem of analytical queries taking several hours to run.
- Goals
  - Optimize Performance Efficiency Index (PEI) to reduce query execution times.
  - Collect experimental data to validate our hypothesis.
  - Help create slides for our client's presentation due at the end of February.

Current State

- Finished with bulk of research
- Key parameters and features identified
  - Leveraged for experimentation
- Collected experimental data overnight
- Finished assisting in presentation preparation
- Wrapping up document revision

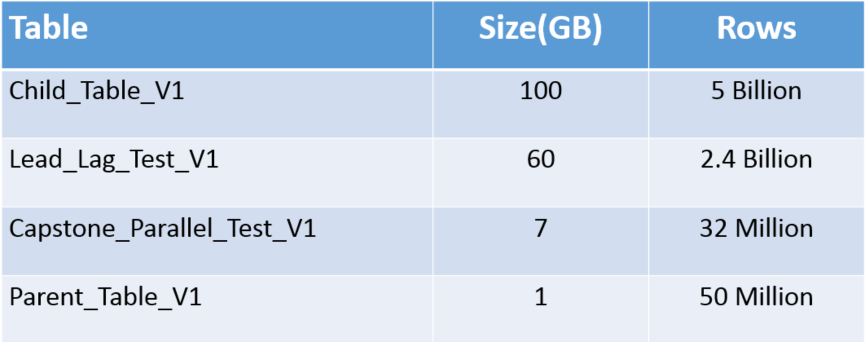


Table	Size(GB)	Rows
Child_Table_V1	100	5 Billion
Lead_Lag_Test_V1	60	2.4 Billion
Capstone_Parallel_Test_V1	7	32 Million
Parent_Table_V1	1	50 Million

Fig 1. Description of SQL test tables

Nic’s Experience

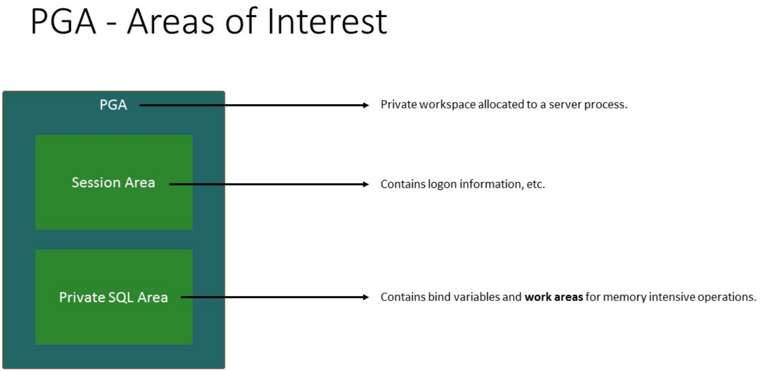
Winter

- Met with client to discuss Winter research topics and expectations.
- Focus on tracing events within the database.
- Create ramdisk for temp tablespace.
- Started web based toolkit.

Week One

- Setup twice weekly meetings for the term.
- Figure out Program Global Area (PGA).
- Research PGA parameters and features.
- Figure out System Global Area (SGA) afterwards.
- Create slides detailing PGA components.

Week One



PGA - Areas of Interest

PGA → Private workspace allocated to a server process.

Session Area → Contains logon information, etc.

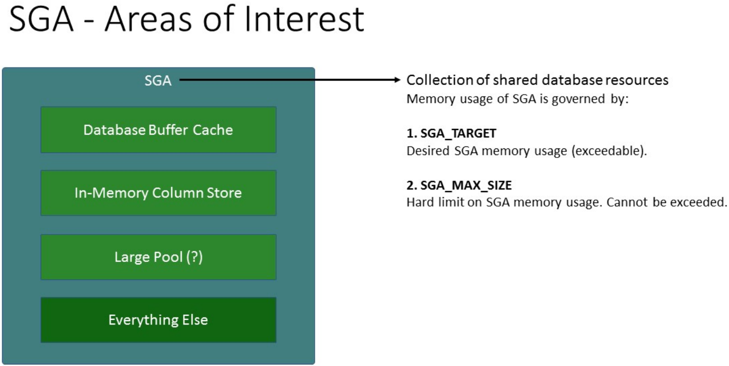
Private SQL Area → Contains bind variables and **work areas** for memory intensive operations.

Fig 2. Structure of PGA for private server process.

Week Two

- Cleaned up slides on PGA.
- Research In-Memory Parallel Execution.
    - Speed up parallel execution.
    - Make use of block cache.
    - Reduce disk I/O.
- Started researching SGA.

Week Two



SGA - Areas of Interest

SGA

Database Buffer Cache

In-Memory Column Store

Large Pool (?)

Everything Else

Collection of shared database resources
Memory usage of SGA is governed by:

1. SGA_TARGET
Desired SGA memory usage (exceedable).

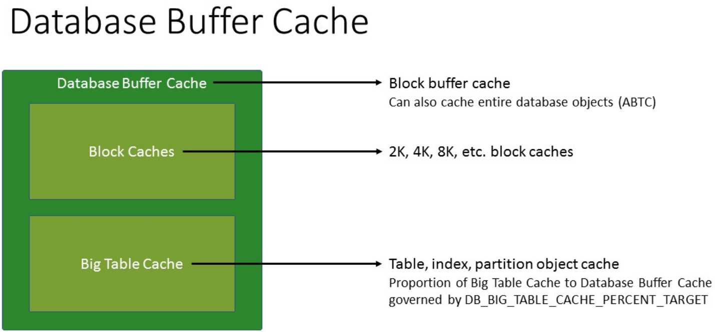
2. SGA_MAX_SIZE
Hard limit on SGA memory usage. Cannot be exceeded.

Fig 3. Partial structure of SGA.

Week Three

- Started detailing SGA components.
- One key component of SGA is the block buffer cache.
  - Used by In-Memory Parallel Execution
- Researched Automatic Big Table Caching.
  - Caches entire database objects (tables, etc.)
  - Complements block buffer cache.
- Researched In-Memory Column Store.
  - Columnar format cache.
  - Benefits analytical queries.

Week Three



Database Buffer Cache

Database Buffer Cache → Block buffer cache

Can also cache entire database objects (ABTC)

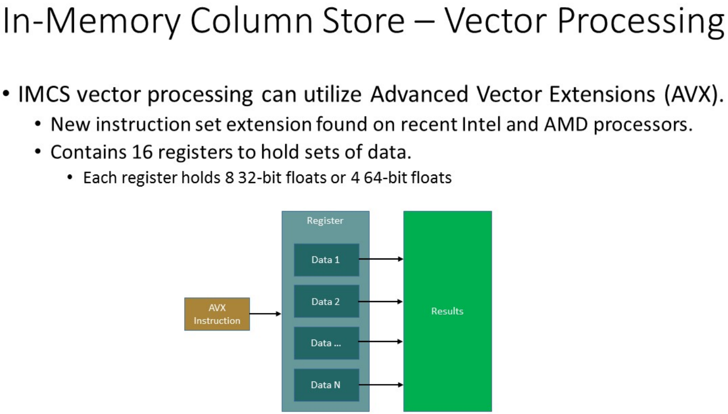
Block Caches → 2K, 4K, 8K, etc. block caches

Big Table Cache → Table, index, partition object cache

Proportion of Big Table Cache to Database Buffer Cache governed by DB_BIG_TABLE_CACHE_PERCENT_TARGET

Fig 4. Structure of Database Buffer Cache within SGA.

Week Three



In-Memory Column Store – Vector Processing

- IMCS vector processing can utilize Advanced Vector Extensions (AVX).
  - New instruction set extension found on recent Intel and AMD processors.
  - Contains 16 registers to hold sets of data.
    - Each register holds 8 32-bit floats or 4 64-bit floats

AVX Instruction
Register
Data 1
Data 2
Data ...
Data N
Results

Fig 5. Visualization of Single Instruction Multiple Data (SIMD).

Week Four

- Further researched In-Memory Column Store.
    - Utilizes Single Instruction Multiple Data instruction sets.
    - Performs vector processing on sets of data.
        - Processes batches of data in a single instruction.
        - Enabled scans of billions of rows/sec per CPU core.
- Decided sufficient research was done on key parameters and features.
- Began focusing on creating test suite to test our ideas.

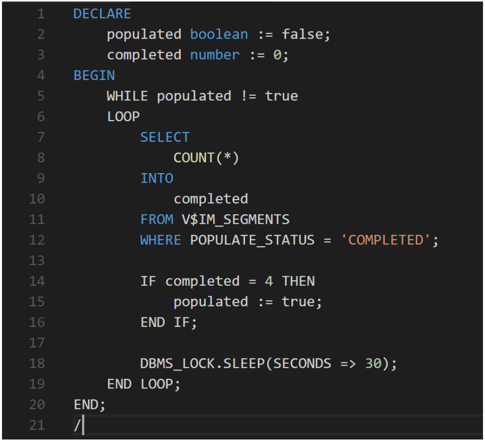
Week Five

- Finished initial test scripts for PGA parameters and Automatic Big Table Caching feature.
- PGA results were promising, enabled more optimal usage of system resources.
- Automatic Big Table Caching results generally propmising, but some unexpected results.
- Began creating In-Memory Column Store test script.
  - Ran into some problems with background processes populating column store.

Week Six

- Created a solution to solve the In-Memory Column Store (IMCS) test problem.
  - PL/SQL script to block IMCS tests from running until column store fully populated.
- However, this had its own problems in testing (path related issue).
- Once fixed, IMCS tests yielded very promising results, significant performance gains.

Week Six



```
DECLARE
    populated boolean := false;
    completed number := 0;
BEGIN
    WHILE populated != true
    LOOP
        SELECT
            COUNT(*)
        INTO
            completed
        FROM V$IM_SEGMENTS
        WHERE POPULATE_STATUS = 'COMPLETED';

        IF completed = 4 THEN
            populated := true;
        END IF;

        DBMS_LOCK.SLEEP(SECONDS => 30);
    END LOOP;
END;
/
```

Fig 6. PL/SQL script to block IMCS tests from running until column store is populated.

Week Seven

- Research was more or less complete at this point.
- So far we were able to tweak database enough to increase performance substantially and resolved some of our initial problems.
- Researched Statement Queueing as finishing touch.
  - Puts queries in queue if optimal amount of resources cannot be given to query (ex: high workload).
  - Led to substantial decrease in overall query processing time.

Nathaniel's Experience

Winter

- Only had one scheduled meeting with client
- Implementing RAM disk as temporary tablespace
- Learning about Angular JS
- Added trace functionality to Toolkit GUI
- Experimented with trace events and external tablespace
- Clients invited to present at Oracle convention: HOTSOS

Week One

- Agreed to two 3-hour meetings with clients each week
- Given individual topics to research:
  - My topic: Extracting clear diagnostic information from trace events
- Categorizing trace events of interest
- Issues with parallel execution event output (ie. 10390/391)
- Reading Oracle white paper on parallel execution
- Started extensive documentation of research

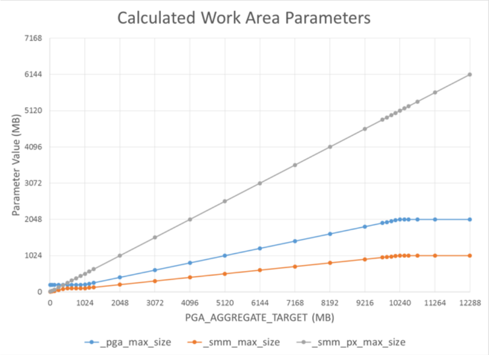
Week Two

- First official term meeting with Jon Dodge
  - Discussed research request from client
  - Jon requested confirmation email from client
  - Brought up troubles with teammate
- Put together OneNote to act as research repository
- Clients requested we use sites labeled Oracle Ace or Oak Table
- False positives with 10390/391 trace output
- Some knobs should remain unturned
  - Bricked database with MEMORY_MAX setting

Week Two Cont.

- Gathered data on three system calculated undocumented parameters
- Goal: Create a mathematical model based on the PGA_AGGREGATE_TARGET value
    - _PGA_MAX_SIZE: Accounting target for a single process
    - _SMM_MAX_SIZE: Max memory allocated to a single work area during serial execution
    - _SMM_PX_MAX_SIZE: Max memory allocated to a single work area during parallel execution

PGA Sizing Calculations



- **_PGA_MAX_SIZE:**

  max (200MB , min (2GB, 20% of PGA_AGGREGATE_TARGET))
- **_SMM_MAX_SIZE:**

  min (20% of PGA_AGGREGATE_TARGET, 50% of _PGA_MAX_SIZE)
- **_SMM_PX_MAX_SIZE:**

  50% of PAT

$$\boldsymbol{Work\ Area} = min \begin{cases} \_SMM\_MAX\_SIZE \\ or \\ \dfrac{\_SMM\_PX\_MAX\_SIZE}{DOP} \end{cases}$$

Fig 7. Visualization of system calculated parameter relationship

Week Three

- Nothing major to report
- Time spent reading through Oracle In-Memory paper
- Things with James continue to grow stressful
  - Lack of communication with us and clients
  - Shows general frustration with feeling behind
  - Never asked Nic or myself for help
- Initial research for PGA experiment completed
- Plan on doing targeted experiments on PGA and Big Table Cache

Week Four

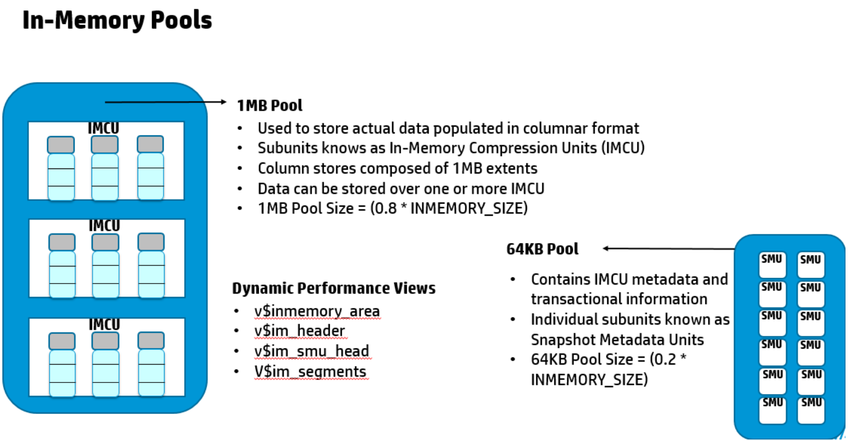
- Began reading Oracle paper "When to use In-Memory"
- Gathered insights that directed our In-Memory experiments
- Started developing test automation suite based on Andy's mental model
- Working prototype by the end of the meeting consisted of:
  - driver.sh
  - runMonitor.sh
  - runTest.sh
- Spent time cleaning up and commenting scripts
- Added remaining work units to GitHub issue tracker

Week Five

- Original design of test automation suite did not work
  - Monitor loop terminated on database restart
  - Resolved by altering the statement execution order within the driver
- Added requested features to driver
  - Loop to run parameter files before SQL test
  - Lockfile to prevent process interrupt
  - Feature to email log file to distribution list
- Started the three main experiments before the end of Fridays meeting

Week Six

- Monitored experiments progress throughout the weekend
- Initial reports seemed promising for each experiment
- Noticed unexpected results from In-Memory experiment
  - Much longer runtimes than expected
  - Cause quickly identified
  - All tables did not fit in IMCS without compression
- Split up In-Memory experiment
- Started slides on IMCS memory pools
  - 1 MB Pool: Stored actual data in columnar format
  - 64 KB Pool: Metadata and transactional data on data stored in 1 MB pool



In-Memory Pools

1MB Pool

- Used to store actual data populated in columnar format
- Subunits knows as In-Memory Compression Units (IMCU)
- Column stores composed of 1MB extents
- Data can be stored over one or more IMCU
- 1MB Pool Size = (0.8 * INMEMORY_SIZE)

Dynamic Performance Views

- v$inmemory_area
- v$im_header
- v$im_smu_head
- V$im_segments

64KB Pool

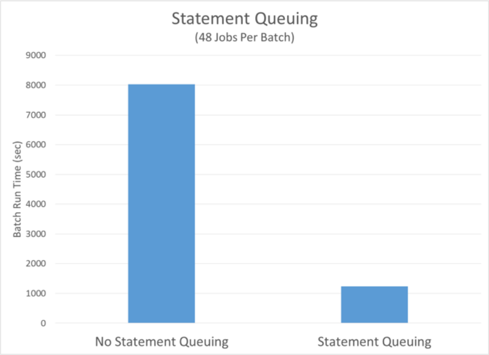
- Contains IMCU metadata and transactional information
- Individual subunits known as Snapshot Metadata Units
- 64KB Pool Size = (0.2 * INMEMORY_SIZE)

Fig 8. Description of IMCS memory pools

Week Seven

- Time spent finalizing slides for clients presentation
- Clients requested an experiment on statement queuing
- Nic and I brainstormed an experimental procedure
- Initial attempt was a failure
    - Used a test that joined 1GB and 100GB table
    - Exhausted available PGA due to numerous concurrent query executions
- Issue resolved by increasing PGA_AGGREGATE_TARGET
- Results clearly demonstrated the cost of resource contention

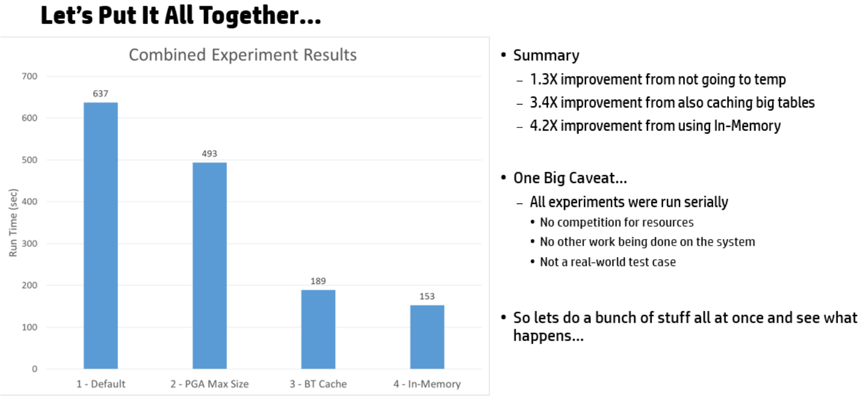
Experiment # 4: Statement Queuing



- Summary
  - Statement queuing works like expected!
  - Without queuing most jobs get massive downgrading and therefore lots of swap to temp
    - Data was In-Memory for both sets of tests.
  - With queuing each job gets the optimal resources it needs to execute
  - Not shown:
    - If you keep in parallel hints and have queuing turned off your queries will start to die with: **ORA-04036: PGA memory used by the instance exceeds PGA_AGGREGATE_LIMIT**

Retrospective

Positives	**Deltas**	**Actions**
Presentation went well	Web based toolkit security	Sanitize input
Promising results on all experiments	Complete web based toolkit	Implement more functionality in web based toolkit
Was able to focus more on project	Roll changes to production	Develop road map with client
Revised design document	-	-
Developed understanding of Oracle database concepts	-	-



Let’s Put It All Together…

- Summary
  - 1.3X improvement from not going to temp
  - 3.4X improvement from also caching big tables
  - 4.2X improvement from using In-Memory
- One Big Caveat…
  - All experiments were run serially
    - No competition for resources
    - No other work being done on the system
    - Not a real-world test case
- So lets do a bunch of stuff all at once and see what happens…

Fig 9. Overview of main experimental results

Conclusion

- Finished all of our desired goals
    - Optimized database for client use cases
- Lost a member of our group
- Clients reported that there presentation went well
    - They were asked to present again next year
- Moving forward:
    - Further flesh out our testing suite
    - Work with clients to roll changes out to production
    - Set production environment up to better handle DSS workloads