CARAVEL

Performance analysis in modernization projects

BASE100

BASE 100, S.A.
www.base100.com

Document: performance_analysis_in_modernization_projects_v3_en.docx
Version: 3.0

BASE 100, S.A.
http://www.base100.com

Contents

Performance in Modernization processes

Overview

The modernization of AS/400[1] legacy systems to open platforms has to solve various problems. Among the most important are those related to performance.

It is well know that AS/400 is a very reliable and extremely high performance platform. Every component of AS/400 has been designed to work together, focusing reliability and performance over standardization or compatibility. So the maintain same level in the new system, it is necessary and intensive use of technology and know-how.

Since year 2000 Caravel has been used in more than 100 modernization projects.

Along this period, the BASE 100 technical team has analyzed the characteristics of many applications from reliability and performance point of view. Modern open systems offer levels of reliability alike to the legacy systems. On the other hand, performance is always an issue that must be considered.

The fast evolution of open systems has increased performance, especially when considered related to costs. This fact and the sharp evolution of Caravel technology have eliminated the prior performance gap. Now, in many cases, the converted systems are faster than the original one, just after an automatic modernization process and configuration tuning. Minor reengineering interventions can also be introduced at any time to improve performance. Only in few cases mayor reengineering or platform modifications are required.

Experience

Find below several tables resuming the experience after all of these years. This is based on Caravel long activity:

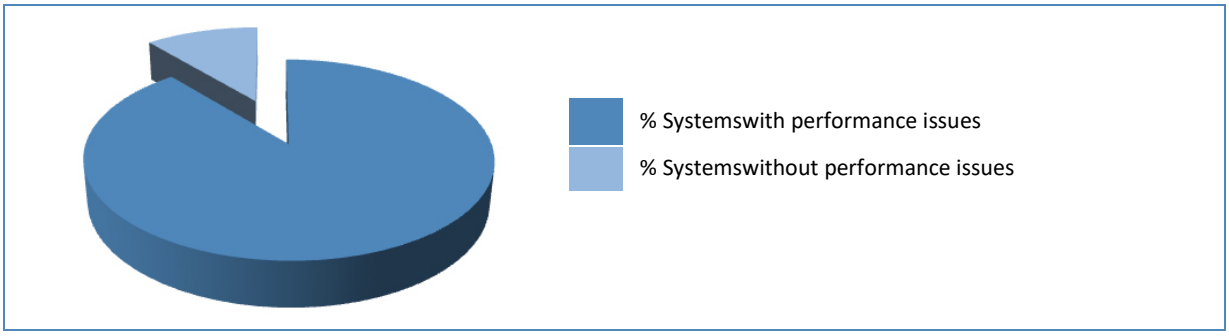
- Number of objects converted by Caravel: 1.440.000.
- Number of code lines converted by Caravel: 936.000.000.
- Data Volume migrated: More than 10.000 GB.

[1] AS/400 means also: S/36, S/38, iSeries, System i.

Statistics

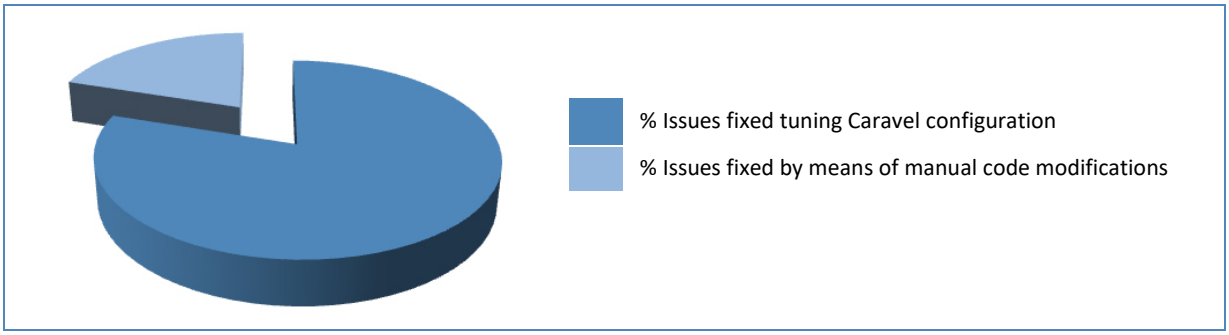
Performance issues

- % Systems with performance issues: 89%.
- % Systems without performance issues: 11%.



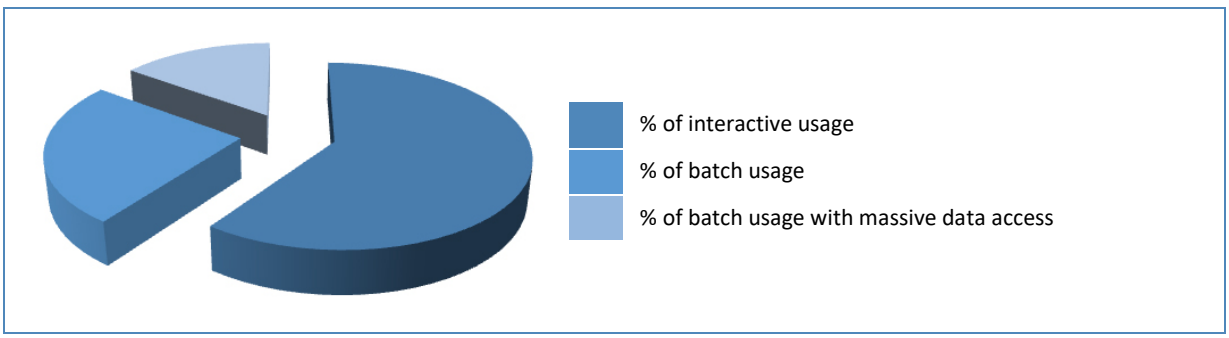
Fixing of performance issues

- % Performance issues fixed automatically tuning Caravel configuration: 80%.
- % Performance issues fixed by means of manual code modifications 20%. (In most cases, minor modifications.)



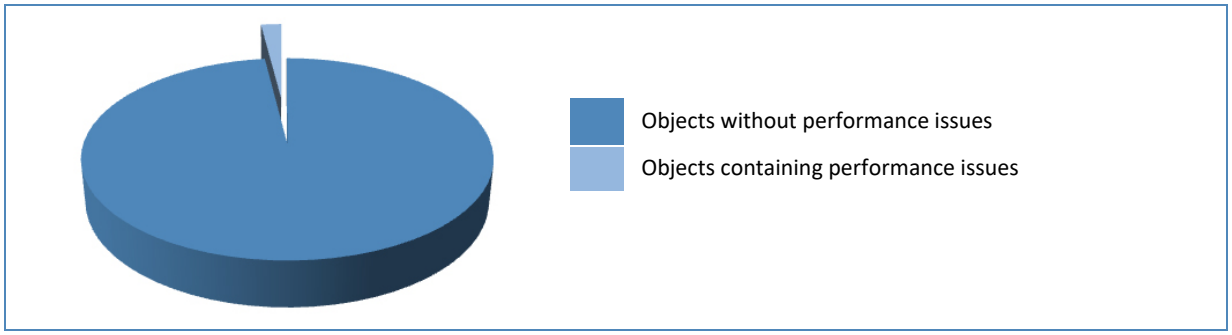
Characteristics of the applications with performance issues

- % of interactive usage: 60%.
- % of batch usage: 25%.
- % of batch usage with intensive data access: 15%



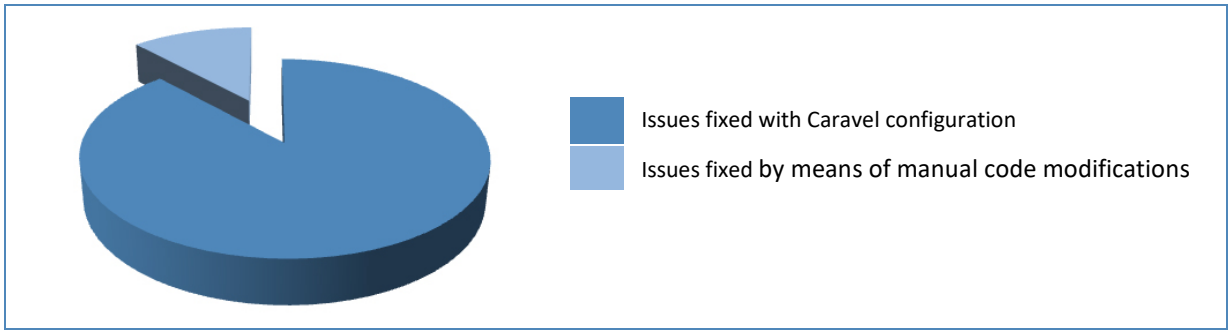
Sample of a real modernization project made by Caravel

- Objects converted without performance issues: 49.635
- Objects converted containing performance issues: 1.009



Sample II of a real modernization project made by Caravel

- Performance issues fixed with Caravel automatic conversion and configuration tunning: 15.
- Performance issues fixed by means of manual code modifications: 2.



Performance possible problems

The performance issues have different causes, find enclosed a detail of the most commonly found.

Massive data access (Cycle programs, OPNQRY and CPYF usage)

AS/400 applications are optimized for a massive “one by one” data treatment; conversely modern applications are oriented and optimized for a SQL data treatment.

This kind of problematic appears basically in the BATCH processes that perform millions of data access (read and write).

By means of the Caravel SEQUENCER tool, is possible to detect where an issue is located and what is the kind of the problem. Additionally Caravel SEQUENCER allows establishing the optimal solution in every case.

Problems with some JDBC drivers: the READP instruction

Due not all of the Java Drivers have the same characteristics, in different projects appear situations where is needed a specific JDBC management. One common situation that affect the performance is the SCROLLABLE CURSOR usage. The READ NEXT + READ PREVIOUS behabiour can be simulated by using scrollable cursors, (if the Database Server allows this feature), but in many cases it may affect the performace results.

When the use of scrollable cursors is a problem, caravel allows by means of configuration, the use of double cursors (ascending / descending) to obtain better performance results. To use this feature, both ascending and descending indexes are required.

> **SAMPLE:**
> With the property “useReverseIndexesByDefault” Caravel configures the doubles indexes usage: ASCCENDING and DESCENDING.
>
> In the indexes process creation for the database, if this property is active, two indexes are created for same columns. One with ASCENDING order and the other one with DESCENDING order.
>
> When one READP instruction is found on the execution of some program, Caravel Framework generates the SELECT instruction using the ASCENDING or DESCENDING index (depending on the case).

SELECT/OMIT treatment in Logical Files

This kind of prebuilt selection conditions are characteristics of the DB2/400 and doesn’t have a equivalent functionality in standard SQL DBMS.

In some cases, the response time in these logical files when using SQL is lower compared with the original AS/400 because the SELECT/OMIT evaluation is made each time it is processed while in DB2/400 this evaluation is made when the records are written.

These problems are analyzed using the Caravel SEQUENCER tool and then is applied a specific solution for every case.

Reengineering methodology for performance with Caravel Sequencer

Below are described the steps followed to optimize the performance with Caravel SEQUENCER.

1. Configure Caravel with "auditDiskFiles" and run the process to be analyzed. This property allows analyzing all of the disk data accesses.
2. Check the result: number of operations READ, WRITE and UPDATE on each file.
3. Analyze the source code to determine what conditions have the read rows, to perform changes.
4. Change the Java class and add the condition, and repeat the process execution.
5. Compare the result of both processes:

Example

- Let’s suppose a batch process with massive data access (read and update). After the first process execution the results are:
  READS: 150.000, UPDATES: 500.
- The updates condition are analyzed: Apparently the rows to be modified, have the following condition: COL = 5
- Then Java classes are changed replacing the “row by row” read by a SQL “where condition” and the process is executed again.
- The process continues updating 500 rows, and only 1500 reads are required.

This reflect common situation in performance-oriented reengineering processes, converting RPG/COBOL code to SQL statements, replacing the massive processes "row by row" with more accurate SQL statements.

General rules to address performance improvement

In terms of performance improvement, Caravel Conversion Projects distinguish three levels to perform different actions.

Level I : Configuration and Tuning

Caravel Framework offers different levels of parameterization to customize each installation. These parameters can improve time execution.

These parameters can control, among other functionality:

- Preload caches.
- Database tables cache (Lambert[2]).
- JOB Queue Servers parallelization.
- Sql Cursors Fetch Buffer size.
- Select Index by Hint.
- Delegate Locking on Caravel System Server.
- Sql Blocking Factor.

Others components like JVM, DBMS Server, Application Server can be also tuned. A set of different aspects for each one can be set in order to improve the application performance.

Level II: Caravel Translator and Framework customization

In special cases, the application performance can be improved by extending some characteristics on the Caravel translators, and/or allowing new features of the Caravel Framework.

Most of the typical improvements have already been considered and are now available on the current Caravel version. But when there is not an existing way to solve a performance issue, it is always considered the possibility of create a new improved version of the Caravel translator or Framework that could solve it.

As a result of this, we could obtain:

- A better generated java code in terms of performance or
- New Framework parameters defined to solve this specific problem.

The current modifications will be included in the next Caravel versions and are available for the following project conversions.

Level 3 Improvement by Reengineering

Finally, if there is not a suitable solution for an application performance issue, we should modify the application code to find a way to get the expected results.

These changes can be applied on:

- Destination. We can obtain these result by changing on the resulting java code (loop optimization, use of native variable instead of RPG/COBOL translated variables, parallelization, etc.)

[2] The Cache Lambert is a specific technology developed by Caravel technicians. It detects repetitive file record reads, and suggests potential data files to be cached in memory to improve applications performances.

- Origin: We can obtain the expected result just making the appropriate changes on the original source code components (programs, files structure or indexes, configuration).

Optimizing by Parallelization

When it is possible determine parallelizable execution blocks, significant improvements can be achieved since the timeouts in Input / Output operations can be used to execute other parallel threads. Every thread waits for execution ending of each of the blocks and consolidates the results, in order to continue the normal program execution.

There are always a maximum number of simultaneous threads that can be parallelized, and can be parameterized in the reengineered program.

Optimizing database access

In all major DBMS (DB2, Oracle, SQL Server, Informix, etc) there are configuration capabilities to optimize database access, reducing costs in I/O operations (and therefore reducing elapsed time). It is mandatory to make some optimization analysis and determine the values of Database configuration parameters. This fine tuning process should involve at least the next actions groups:

- Optimize by caching in memory tables that are read many times but rarely modified
- Using Clustered Index and Partitions in large tables that always are read in the same order.
- Using SQL pattern detection (and hints) to pre-compile query plans of very I/O heavy queries in order to use specific indexes.
- Partition data storage to place the most accessed tables in the most appropriated device.
- Use of forward only cursors for processes that never read records in reverse order.