# Instituto Tecnológico y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Department of Mathematics and Physics

**Master of Data Science**



## Clustering Models for Analyzing the Database Update Process

---

**RECEPTIONAL WORK** to obtain the **DEGREE** of
**MASTER OF DATA SCIENCE**


Presented by:
**Pablo Ignacio González Rivera**

Advisor:
**MS. Byron Michael Motta Bonilla**


Tlaquepaque, Jalisco, December 16, 2025

# Clustering Models for Analyzing the Database Update Process

Pablo Ignacio González Rivera

## Abstract

*This work presents an empirical characterization of Oracle Datapatch execution during live patching by analyzing performance telemetry generated by the Oracle RDBMS. The main problem addressed is the lack of studies, datasets or methodologies describing how Datapatch behaves under real runtime conditions. The general objective of this work is to model this behavior using unsupervised learning techniques in order to identify recurring execution patterns.*

*To achieve this objective, a data acquisition strategy based solely on native and license free instrumentation was developed using Statspack. A structured feature selection process was then applied using two complementary approaches: a statistical method based on a supervised model to identify variables correlated with execution duration, and a semantic method based on Oracle documentation to construct interpretable performance indicators. Both variable sets were used to train and evaluate clustering models capable of grouping similar execution behaviors.*

*The results show that the proposed methodology can distinguish between stable executions, executions with internal pressure spikes handled efficiently and executions characterized by dominant I/O demand. These clusters provide insight into runtime behavior beyond elapsed time measurement and demonstrate that Datapatch performance varies across identifiable operational states. The main contribution of this work is the development of a reproducible methodology for understanding Datapatch execution behavior through data driven analysis. Finally, the conclusions and potential research extensions are presented.*

# Clustering Models for Analyzing the Database Update Process

Pablo Ignacio González Rivera

## Resumen

*Este trabajo presenta una caracterización empírica del comportamiento de Oracle Datapatch durante su ejecución en línea mediante el análisis de métricas de desempeño generadas por el propio motor de base de datos. El problema principal que se aborda es la ausencia de estudios, metodologías y conjuntos de datos que describan cómo se comporta Datapatch en tiempo real, especialmente bajo diferentes condiciones de carga y variabilidad operativa. El objetivo general de este trabajo consiste en modelar dicho comportamiento empleando técnicas de aprendizaje no supervisado para identificar patrones recurrentes durante la ejecución del parcheo SQL.*

*Para alcanzar este objetivo, se desarrolló una estrategia de adquisición de datos basada en Statspack, seleccionando únicamente instrumentación nativa y libre de licenciamiento adicional. Posteriormente se aplicó un proceso de selección y construcción de variables utilizando dos enfoques complementarios: uno estadístico mediante un modelo supervisado y otro semántico basado en documentación de Oracle. Con estas variables se entrenaron modelos de clustering para identificar patrones de ejecución y validar su relevancia operativa.*

*Los resultados muestran que los modelos obtenidos permiten distinguir entre ejecuciones estables, ejecuciones con actividad interna elevada pero controlada y escenarios con presión dominada por I/O. La principal aportación de este trabajo es la creación de la primera metodología reproducible para analizar el desempeño de Datapatch en tiempo real. Finalmente, se presentan las conclusiones del estudio y posibles líneas de trabajo futuro.*

# Contents

# List of Figures

**Página**

# List of Tables

**Página**

*Dedicado a mis abuelos, Sofy, Papeto y Lulú, que me siguen bendiciendo.*

*A Paty, mi madre, maestra y ejemplo a seguir.*

*A Javier, mi padre, por enseñarme lo que es el ingenio.*

*A Javi y Felipe, mis hermanos, que abrieron caminos para que yo pudiera correr.*

*A Emi y Andrés, mis sobrinos, que son el futuro y la esperanza.*

*A Itzel, mi novia, quien con su paciencia y amor me sostuvo incluso cuando yo dudaba.*

*A Alejandro, mi amigo, por ser mi brújula y empuje aun cuando los días eran demasiado largos.*

# 1 Introduction

## 1.1 Context

*Oracle RDBMS patching consists of two primary components: Binary Patching and SQL Patching.*

*Binary Patching updates the Oracle Home to a target version and relinks all required executables, allowing the database to operate with the latest binary codebase. SQL Patching, in contrast, applies the necessary changes within the database itself, including updates to Oracle-maintained objects and data dictionary structures to maintain consistency with the patched Oracle Home.*

*Unlike database upgrades or migrations, patching is a highly recurrent operational activity. Oracle delivers consolidated sets of recommended fixes on a quarterly basis through Release Updates (RUs), leading most production environments to execute patching procedures multiple times per year.*

*Modern production deployments increasingly perform RUs in online mode, allowing SQL Patching to occur while applications remain connected and workloads continue to execute. This makes patching not merely a maintenance task but a live operational event that unfolds within an active system under real transactional load.*

*During SQL Patching, the database operates as a fully functional system, where background processes, user transactions, shared memory structures, and concurrency mechanisms remain active. As a result, the behavior of the patching process emerges from a dynamic environment shaped by internal performance signals and runtime conditions. Accurately characterizing this environment requires observing the database in situ during patch execution, rather than relying solely on isolated or synthetic test scenarios.*

## 1.2 Justification

*This research investigates Oracle RDBMS behavior during live patching operations using Statspack, a native, vendor supported, and license free performance instrumentation framework. Although patching is a routine and critical operational activity, the internal runtime conditions under which SQL patching executes remain poorly characterized in a systematic, data-driven manner.*

*By collecting operational telemetry, applying feature selection to isolate statistically and operationally relevant variables, and using unsupervised clustering techniques, this study seeks to identify recurring patterns in database behavior during patch execution. The findings aim to establish a data-driven taxonomy of patching environments, enabling improved operational predictability and contributing to empirical research on live database system behavior.*

## 1.3 Problem

### 1.3.1 General Problem

*Despite the routine and critical role of Oracle RDBMS patching in enterprise environments, there is limited systematic understanding of how live database operational conditions influence the behavior and performance of SQL patch execution.*

*Patching is typically evaluated based on success or failure outcomes, while the internal runtime environment in which patching occurs remains largely uncharacterized in a structured, data-driven manner.*

### 1.3.2 Scientific Problem

*There is currently no data-driven framework for modeling and classifying database operational states during live patching events, nor for identifying which internal performance variables most significantly shape patch execution behavior.*

*This limits the ability to empirically analyze, compare, and predict the impact of different workload and system conditions on patching dynamics using reproducible, unsupervised analytical methods.*

## 1.4 Objetives

### 1.4.1 General Objective

*To empirically characterize and model Oracle RDBMS operational behavior during live SQL patching by analyzing performance telemetry and applying unsupervised learning techniques to identify recurring behavioral patterns.*

### 1.4.2 Specific Objetives

1. *To design a data acquisition strategy based on native, license free instrumentation capable of capturing operational metrics during live patching events.*
2. *To formalize a feature engineering and selection process for identifying performance variables with significant influence on patch execution behavior.*
3. *To implement and evaluate clustering models for the identification of recurrent behavioral patterns in patching environments.*
4. *To establish interpretability criteria for validating the operational relevance of the discovered clusters.*

# 2 Methodology

*In this chapter, the methodological development is presented in detail, including the creation of the data set using Oracle Statspack, the data cleaning process, the selection of variables, the choice of models and the metrics used for evaluation.*

## 2.1 Dataset Description

*When the planning phase of this project began it was examined whether any previous studies addressing the performance behavior of Datapatch had been conducted. No documented research or analysis was found. Consequently this work is the first effort to extract performance data from Datapatch in order to enable its later use in data science models.*

*Then, reviewing the tools that Oracle provides for evaluating database performance, several enterprise level options were considered, including the Automatic Workload Repository (AWR) [1], Active Session History (ASH), and Statspack. After analyzing their characteristics, Statspack was identified as the most appropriate choice. It does not require any additional license, it is available at no cost in every Oracle edition, and all the elements needed for its installation are already included in the Oracle Home environment. As a result, the installation process is simple and can be completed through a small number of clearly defined steps.*

*Oracle Statspack [2] is a diagnostic framework designed to provide a systematic and historical perspective on database performance. It works by taking periodic snapshots that record cumulative metrics from internal Oracle components such as wait events, latch behavior, buffer cache activity and SQL execution patterns. These snapshots are stored in the PERFSTAT schema and later compared to obtain performance deltas, which reveal what happened specifically between two points in time. Through this method, Statspack offers a structured and reliable way to understand workload characteristics, identify bottlenecks and support informed decision making during performance tuning.*

*The following stage focused on extracting the information required for the project. The proposed approach consisted of installing Statspack [3] in each*

*pluggable database within the selected environments. Once the installation was completed, an initial snapshot was taken, followed by the execution of Datapatch. After the procedure concluded, a final snapshot was captured. With this sequence in place, the performance data generated during the Datapatch execution became fully available in the Statspack tables, which allowed the information to be queried and analyzed in any manner needed for the study.*

*With this strategy in mind, a custom script was developed to execute the sequence required to generate the initial raw data set:*

1. *Ensure that Statspack is properly installed in each of the pluggable database.*
2. *Capture the initial snapshot before the Datapatch execution.*
3. *Execute the Datapatch script with the selected parameters.*
4. *Capture the final snapshot after the Datapatch execution.*
5. *Retrieve the Statspack data and generate the corresponding reports in comma separated values format.*

*The custom script was executed in several environments with different configurations, and all the resulting reports were later consolidated into a single file. This initial data set contained all the information retrieved from the Statspack tables during the execution of Datapatch. It consisted of 144 observations and 1321 variables. This dimensionality was later reduced through a series of data cleaning procedures and a structured process of variable selection.*

## 2.2 Exploratory Data Analysis

*Because one of the objectives of this project is to implement and evaluate clustering models, and because the initial raw dataset contained a large number of variables, a structured data cleaning procedure was performed to reduce the dimensionality and prepare the dataset for further analysis.*

*The initial shape of the dataset was 144 rows and 1321 columns. The following transformations were then applied sequentially to refine and prepare the data.*

1. *Selected only the numerical variables.*
2. *Removed columns that contain any null value.*
3. *Removed columns with constant values.*
4. *Removed columns (after MinMaxScaler was applied) where variance was lower than 0.05.*

5. *Remove columns where more than 95% of the values were zeros.*

*After applying these filtering steps, the dimensionality of the dataset was reduced to 144 rows and 746 columns, Although this represented a significant reduction, the number of variables remained too high to apply clustering directly in a meaningful or computationally practical way. It was therefore necessary to define a structured approach for selecting a smaller and more informative subset of variables for the clustering stage. Two alternative strategies were considered:*

1. *Selecting variables based exclusively on statistical or data science criteria.*
2. *Selecting variables based on semantic relevance and the information provided by the Statspack documentation.*

*Each of these strategies offers different advantages and considerations, and their detailed analysis is presented in the following subsections.*

### 2.2.1 XGBoost Variable Selection

*After applying the initial filtering steps to reduce incomplete and uninformative columns, an XGBRegressor model was executed using runtime_minutes as the target variable. Because XGBoost requires a supervised target, runtime_minutes was selected as it represents the total duration of the Datapatch execution and provides a meaningful quantitative reference for performance behavior. The purpose of this stage was not to construct a predictive model, but to identify which variables contribute most strongly to the variation in elapsed time.*

*XGBRegressor [4] is a gradient boosting algorithm that constructs an ensemble of decision trees, where each tree iteratively reduces the residual errors of the previous ones. It is well suited for this type of task because it captures complex non linear interactions, handles large numbers of candidate variables and generates interpretable importance scores that reflect the contribution of each feature to the model. These properties make it appropriate for dimensionality reduction in a dataset of shape (144, 746).*

*Using XGBoost for variable selection has two main advantages in this context. First, its feature importance scores allow the identification of the attributes most associated with the duration of Datapatch execution. Second, its regularization mechanisms help mitigate overfitting even in high dimensional settings. However, it is important to acknowledge two limitations of this method. Because the target variable is runtime_minutes, the selected variables are naturally biased toward time related behavior, and therefore may not fully reflect broader internal performance dynamics. Additionally, although statistically informative, these variables originate from internal*

*Statspack counters whose documentation is limited, which constrains their interpretability.*

| Feature | Importance |
|---|---|
| BODY_DELTA_PINHITS | 0.340672 |
| enqueue hash chains_DELTA_GETS | 0.147652 |
| CPU used when call started_DELTA_VALUE | 0.091131 |
| non-idle wait count_DELTA_VALUE | 0.072290 |
| dc_segments_DELTA_GETS | 0.064093 |

Table 2.1: Feature importance values for selected performance variables.

*Despite these constraints, the method produced a compact and informative subset of five variables 2.1, which were renamed and represent the attributes with the strongest association to elapsed time:*

- *plsql_body_pinhits*
- *enqueue_latch_gets*
- *cpu_call_time*
- *non_idle_waits*
- *dc_segments_gets*

*This selection yielded a reduced dataset of shape (144, 5), containing only numeric values and no missing entries. A descriptive summary of the reduced dataset is provided in Figure 2.1.*

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| plsql_body_pinhits | 144.0 | 28703.277778 | 21937.413150 | 4650.0 | 8489.25 | 26730.0 | 43278.00 | 90518.0 |
| enqueue_latch_gets | 144.0 | 835166.680556 | 631012.552727 | 214892.0 | 300093.75 | 597812.5 | 1158179.00 | 2916979.0 |
| cpu_call_time | 144.0 | 73453.555556 | 57424.461917 | 11302.0 | 39328.75 | 53859.0 | 98610.00 | 240601.0 |
| non_idle_waits | 144.0 | 106483.888889 | 90318.212712 | 5129.0 | 46623.50 | 90841.5 | 137569.75 | 362160.0 |
| dc_segments_gets | 144.0 | 17646.791667 | 11344.916327 | 1723.0 | 10489.75 | 15899.0 | 21751.25 | 47694.0 |

Figure 2.1: Dataset description

*The correlation matrix of the selected variables is presented in Figure 2.2, and their distributions prior to standardization are shown in Figure 2.3. These visualizations provide an initial view of their relationships and scale differences before their later integration into the modeling stage.*

*This reduced and well structured set of variables will be used in the next stage of the methodology, where clustering models are applied to examine potential groupings within the Datapatch executions.*

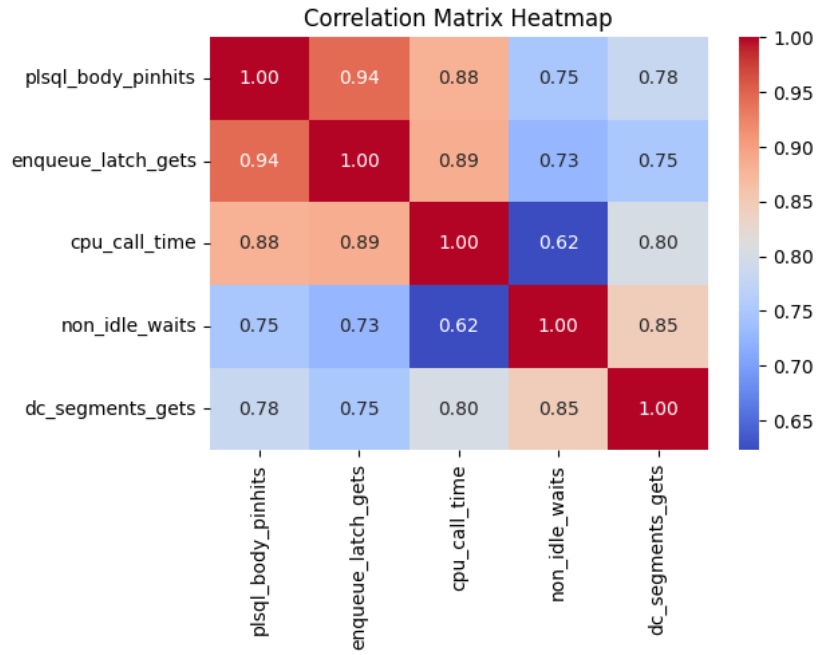


Figure 2.2: Correlation matrix of the XGBoost selected variables

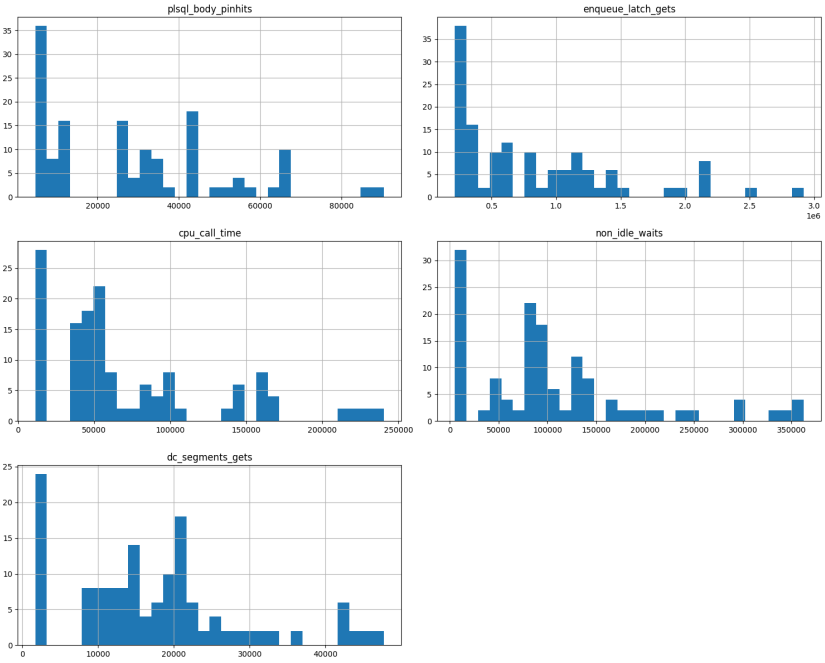


Figure 2.3: Distribution of the XGBoost selected variables

### 2.2.2 Semantic Variable Selection Strategy

*To complement the statistical feature selection performed with XGBRegressor, a semantic strategy was developed to reorganize raw Statspack counters into higher level performance indicators. Because the Statspack documentation provides only limited descriptions of the underlying metrics, the conceptual categories presented in the Oracle appendix on common database bottlenecks [5] were used as a structural guide. This document outlines typical subsystem stress areas such as enqueue contention, I/O delays, CPU consumption, buffer cache efficiency, and parsing activity. Based on these categories, a set of engineered "pressure variables" was constructed to summarize each dimension. Simplified symbolic notation is used in the equations below to improve clarity and readability.*

**1. Enqueue Pressure**

$$P_e = \frac{E}{\max(T, 1)} \tag{2.1}$$

*where $E$ denotes enqueue wait events and $T$ is the elapsed time in minutes. This measures contention normalized by execution duration.*

**2. I/O Pressure**

$$P_{io} = \frac{I_s + I_u}{\max(W_s + W_u, 1)} \tag{2.2}$$

*where $I_s$ and $I_u$ are system and user I/O times, and $W_s$ and $W_u$ the respective wait counts. The ratio approximates average time per I/O wait.*

**3. CPU Pressure**

*Each CPU related metric was normalized using:*

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

*and combined into a composite index:*

$$P_{cpu} = 0.25\,C + 0.20\,R + 0.20\,P + 0.15\,E + 0.20\,U \tag{2.3}$$

*where $C$ is normalized DB CPU time, $R$ CPU rate, $P$ parsing CPU time, $E$ parsing elapsed time, and $U$ CPU percentage. The weights balance the contribution of each indicator.*

**4. Buffer Pressure**

$$P_b = S_c + S_l + (100 - H_b) \tag{2.4}$$

*where $S_c$ and $S_l$ are buffer chain and LRU chain sleeps, and $H_b$ is the buffer hit ratio. Higher values indicate reduced buffer cache efficiency.*

**5. Parse Pressure**

$$P_p = 0.25\,H + 0.20\,C_p + 0.20\,N + 0.15\,E_p + 0.10\,Q \tag{2.5}$$

*where $H$ is normalized hard parses per second, $C_p$ hard parse count, $N$ parsing CPU time, $E_p$ parsing elapsed time, and $Q$ percentage of DB time spent parsing. This summarizes parsing intensity during Datapatch.*

*These engineered variables reorganize the Statspack metrics into interpretable representations of enqueue contention, I/O activity, CPU load, buffer efficiency, and parsing behavior. While the formulas are not defined explicitly in Oracle documentation, their construction is guided by the subsystem categories described in the bottleneck appendix, providing semantically meaningful structures for downstream clustering analysis.*

*The correlation matrix obtained from these engineered variables is presented in Figure 2.4, and the distribution of each variable prior to standardization is shown in Figure 2.5.*

## 2.3 Model description

*To analyze the dataset a grid style methodology was adopted. This approach consisted in applying several clustering models and evaluating their performance across a predefined range of two to six clusters. The objective was to identify stable and meaningful groups that reflect the internal behavior of the Oracle Database during SQL patching operations. Since this workload involves complex interactions inside the RDBMS engine, including latch activity, memory structures, parsing dynamics, and concurrency mechanisms, a comparative evaluation across different clustering strategies was necessary.*

### 2.3.1 KMeans

*KMeans [6, 7] is a partitioning algorithm that groups a set of observations into K clusters by minimizing the total squared distance between each observation and the centroid of its assigned cluster. Expressed as Equation 2.6.*

$$J = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2 \tag{2.6}$$

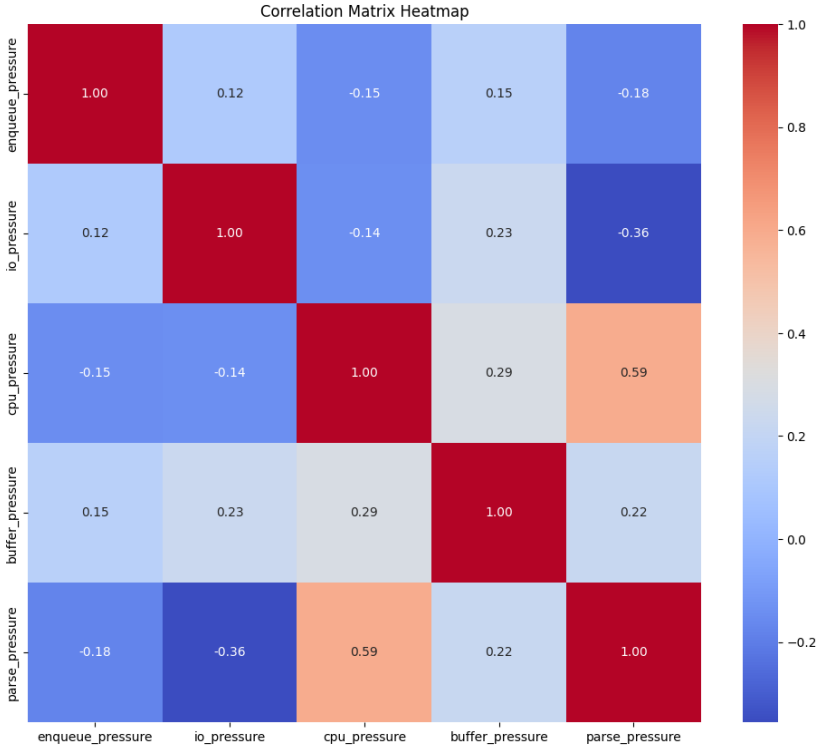


Figure 2.4: Correlation matrix of the engineered variables obtained through the semantic strategy

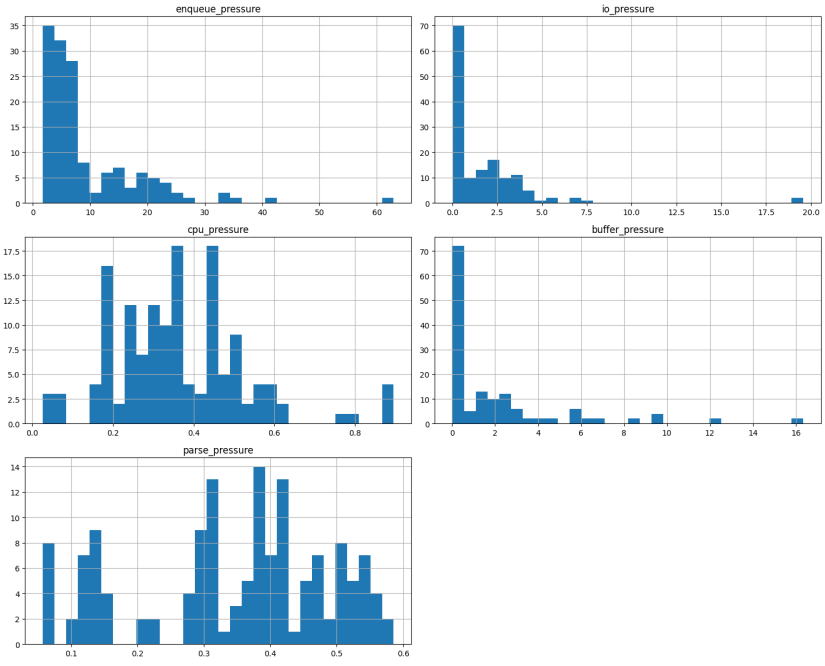


Figure 2.5: Distribution of the semantically selected variables

*where $\mu_k$ represents the centroid of cluster $C_k$. In simpler terms, the algorithm attempts to make each cluster as compact as possible by reducing the squared distances between all points in a cluster and the average location of those points.*

*The centroid of each cluster is computed as the mean of the observations assigned to it:*

$$\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i.$$

*This indicates that a centroid is simply the average position of all points in the cluster.*

*KMeans iterates through two main steps. First, observations are assigned to the cluster whose centroid is closest according to Euclidean distance:*

$$C_k = \{x_i : \|x_i - \mu_k\| \leq \|x_i - \mu_j\| \; \forall j\} \tag{2.7}$$

*Second, once the assignments are updated, centroids are recalculated using the Equation 2.7. These two steps repeat until the assignments stabilize or the objective function shows minimal improvement.*

*For this study, KMeans offers a suitable baseline because the feature selection stage reduces dimensionality and removes variables that could distort distances. Under these conditions, KMeans operates efficiently and provides an interpretable segmentation of the dataset that supports comparison against more advanced clustering techniques.*

### 2.3.2 Spectral Clustering

*Spectral Clustering [8, 9, 10] is a graph based clustering method that groups observations by analyzing the structure of a similarity graph rather than relying on geometric distance in the original feature space. Given a dataset $X = \{x_1, x_2, \ldots, x_n\}$, a similarity matrix $W$ is constructed, where each entry $W_{ij}$ measures the affinity between observations $x_i$ and $x_j$. A common formulation uses a Gaussian kernel, expressed as Equation 2.8.*

$$W_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right) \tag{2.8}$$

*The connectivity of the graph is captured through the normalized Laplacian matrix:*

$$L = I - D^{-1/2} W D^{-1/2},$$

*where $D_{ii} = \sum_j W_{ij}$. Spectral Clustering computes the eigenvectors of L and forms a reduced representation using the first K eigenvectors:*

$$U = [u_1, u_2, \ldots, u_K].$$

*Each row of U represents an observation in a space that reflects the connectivity structure of the data rather than the original feature values. A standard clustering algorithm, typically KMeans, is then applied to this representation to obtain the final clusters.*

*This method is suitable for the present study because it can capture complex relationships among performance related variables and provide an alternative view of execution behavior during SQL patching that complements centroid based approaches.*

### 2.3.3 Ward Hierarchical Clustering

*Ward Hierarchical Clustering [11, 12] is an agglomerative clustering method that builds clusters by progressively merging groups in a manner that minimizes the increase in within cluster variance. Starting from individual observations, the algorithm iteratively merges the pair of clusters whose combination produces the smallest increase in total dispersion. This criterion can be expressed as the change in within cluster sum of squares, expressed as Equation 2.9:*

$$\Delta(C_i, C_j) = \frac{|C_i| \cdot |C_j|}{|C_i| + |C_j|} \|\mu_i - \mu_j\|^2 \tag{2.9}$$

*where $C_i$ and $C_j$ denote two candidate clusters, $|C|$ represents cluster size, and $\mu$ denotes the centroid of each cluster. At each iteration, the pair with the minimal value of $\Delta(C_i, C_j)$ is selected for merging.*

*This iterative process produces a hierarchical structure that represents all possible groupings of the data, ranging from fine grained partitions with many small clusters to coarse partitions with fewer and broader clusters. The resulting hierarchy can be visualized using a dendrogram, which illustrates how clusters evolve as the merging process progresses.*

*This approach is relevant to the present study because performance features collected during Datapatch execution may exhibit nested or layered patterns that are not immediately apparent. When the appropriate number of clusters is not known in advance, the hierarchical structure produced by Ward's method provides a flexible framework for exploring different levels of granularity and identifying distinct workload behaviors.*

### 2.3.4 Gaussian Mixture Models

*Gaussian Mixture Models [13, 14] provide a probabilistic approach to clustering by representing the data as a weighted combination of multiple Gaussian distributions. Each cluster is modeled as a Gaussian component, and the overall data distribution is expressed as a mixture, defined as Equation 2.10:*

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x \mid \mu_k, \Sigma_k) \tag{2.10}$$

*where $\pi_k$ denotes the mixing weight of component $k$, $\mu_k$ is the mean vector, $\Sigma_k$ is the covariance matrix, and $\mathcal{N}(\cdot)$ represents the multivariate normal distribution. The mixing weights satisfy $\sum_{k=1}^{K} \pi_k = 1$.*

*Cluster assignment is performed probabilistically by estimating the posterior probability that an observation belongs to each Gaussian component. This responsibility is computed as Equation 2.11:*

$$\gamma_{ik} = \frac{\pi_k \mathcal{N}(x_i \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i \mid \mu_j, \Sigma_j)} \tag{2.11}$$

*where $\gamma_{ik}$ represents the probability that observation $x_i$ belongs to cluster $k$. Model parameters are estimated using the Expectation Maximization algorithm, which alternates between updating these probabilities and refining the Gaussian parameters.*

*This approach is relevant to the present study because performance related variables collected during Datapatch execution may reflect overlapping behaviors across different execution profiles. Gaussian Mixture Models can capture these shared characteristics and explicitly represent uncertainty in cluster assignment, offering a more nuanced view of workload patterns than hard assignment clustering methods.*

### 2.3.5 MeanShift

*MeanShift [15, 16, 17] is a non parametric clustering algorithm that identifies groups by locating regions of high data density in the feature space. Rather than requiring a predefined number of clusters, the method estimates the underlying probability density function and iteratively shifts observations toward local maxima of this density. The density at a point is commonly estimated using a kernel function, expressed as Equation 2.12:*

$$\hat{f}(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right) \tag{2.12}$$

*where $K(\cdot)$ denotes a kernel function, $h$ is the bandwidth parameter that controls the smoothing scale, $d$ is the dimensionality of the data, and $x_i$ represents individual observations.*

*Based on this density estimate, MeanShift updates each observation by moving it toward the weighted mean of nearby points, which corresponds to the direction of maximum density increase. This update step is defined as Equation 2.13:*

$$x^{(t+1)} = \frac{\sum_{i=1}^{n} x_i K\left(\frac{x^{(t)} - x_i}{h}\right)}{\sum_{i=1}^{n} K\left(\frac{x^{(t)} - x_i}{h}\right)} \tag{2.13}$$

*where $x^{(t)}$ denotes the position of an observation at iteration $t$. As iterations progress, observations converge to density maxima, and clusters emerge naturally as groups of points that share the same mode.*

*This approach is relevant to the present study because it allows the analysis of Datapatch performance data without requiring prior knowledge of the number of execution patterns. When the optimal number of clusters is not evident, MeanShift provides a flexible mechanism for discovering naturally occurring workload structures based on the intrinsic density of the performance variables.*

### 2.3.6 DBSCAN

*DBSCAN [18, 12] is a density based clustering algorithm that forms groups by identifying regions in which observations are densely packed together. Unlike partitioning methods, DBSCAN does not require a predefined number of clusters and explicitly distinguishes between dense regions and sparse areas. This distinction allows the method to identify core observations that define clusters and to label isolated points as noise.*

*The algorithm relies on two parameters, a neighborhood radius $\varepsilon$ and a minimum number of observations MinPts. For a given observation $x_i$, its $\varepsilon$ neighborhood is defined as Equation 2.14:*

$$N_\varepsilon(x_i) = \{x_j \mid \|x_j - x_i\| \leq \varepsilon\} \tag{2.14}$$

*An observation is classified as a core point if the number of observations within its neighborhood satisfies the condition expressed as Equation 2.15:*

$$|N_\varepsilon(x_i)| \geq \text{MinPts} \tag{2.15}$$

*Clusters are formed by connecting core points whose neighborhoods overlap, while observations that do not meet this density requirement are treated as noise and excluded from any cluster.*

*This approach is particularly relevant to the present study because Datapatch execution behavior may include anomalous performance events*

*caused by unusual internal interactions or transient system conditions. DBSCAN enables the separation of these atypical executions from dominant workload patterns, providing a clearer and more robust characterization of meaningful performance groupings within the dataset.*

## 2.4 Description of the Metrics

*The results produced by each clustering method and iteration are evaluated using three internal validation metrics: the Silhouette Score, the Calinski Harabasz Index, and the Davies Bouldin Index. These measures provide complementary perspectives on cluster separation, compactness, and overall structure, allowing a consistent and objective comparison among the different clustering models applied in this study.*

### 2.4.1 Silhouette Score

*The Silhouette Score [19] evaluates how similar an observation is to the other points in its assigned cluster compared to points in the nearest alternative cluster. For an observation $i$, let $a(i)$ denote the average distance between $i$ and all other points in its own cluster, and let $b(i)$ represent the smallest average distance between $i$ and the points in any other cluster. The Silhouette value for $i$ is defined in Equation 2.16.*

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{2.16}$$

*The overall Silhouette Score is obtained by averaging $s(i)$ across all observations. The score ranges from $-1$ to $1$. Values close to $1$ indicate compact and well separated clusters, values near $0$ suggest ambiguous or overlapping boundaries, and negative values indicate that some observations may have been misclassified. This metric provides a direct assessment of cluster cohesion and separation in the present study.*

### 2.4.2 Calinski Harabasz Index

*The Calinski Harabasz [20] Index evaluates the quality of a clustering solution by comparing the dispersion between clusters with the dispersion within clusters. Let $k$ denote the number of clusters, $n$ the total number of observations, $B_k$ the between-cluster dispersion matrix, and $W_k$ the within-cluster dispersion matrix. The index is defined in Equation 2.17.*

$$CH = \frac{\mathrm{tr}(B_k)}{\mathrm{tr}(W_k)} \cdot \frac{n-k}{k-1} \tag{2.17}$$

*Higher values indicate better defined clusters, meaning that cluster centers are well separated and observations within each cluster are relatively compact. This metric is particularly relevant in this study because it measures how effectively the selected Datapatch performance variables distinguish different workload patterns.*

### 2.4.3 Davies Bouldin Index

*The Davies Bouldin Index [21] evaluates the quality of a clustering solution by measuring the average similarity between each cluster and its most similar counterpart. For each cluster i, let $S_i$ represent the average distance between the points in the cluster and its centroid. For two clusters i and j, let $M_{ij}$ denote the distance between their centroids. The similarity measure between the two clusters is given by $\frac{S_i+S_j}{M_{ij}}$. The Davies Bouldin Index is defined in Equation 2.18.*

$$DBI = \frac{1}{k}\sum_{i=1}^{k}\max_{j\neq i}\left(\frac{S_i + S_j}{M_{ij}}\right) \tag{2.18}$$

*Lower values of the index indicate better clustering solutions, since they reflect compact clusters that are well separated from one another. This metric is particularly useful in the present study because it provides a robust assessment of how well the selected Datapatch performance variables form distinct and meaningful workload patterns.*

### 2.4.4 Summary

*Among the internal validation metrics used in this study, the Silhouette Score plays a central role because it directly evaluates cluster cohesion and separation at the level of individual observations. Its ability to measure how well each point fits within its assigned cluster, compared to alternative clusters, provides an intuitive and robust indicator of the overall quality of a partition. This makes it the primary metric for assessing the structure of the clusters obtained from the Datapatch performance data.*

*The Calinski Harabasz Index and the Davies Bouldin Index complement this perspective. The former quantifies the ratio between between-cluster and within-cluster dispersion, while the latter evaluates the similarity between each cluster and its most similar neighbor. Together, these additional metrics offer supporting evidence that strengthens the interpretation of the clustering results. However, the Silhouette Score remains the foremost criterion for evaluating the clustering solutions in this project.*

## 2.5 Description of the Experiments or Simulations

*Two independent clustering experiments were conducted in this study, each using a different set of variables and pursuing a distinct analytical objective. The variables were not mixed at any point in the analysis. Instead, each experiment was executed, evaluated and interpreted separately.*

### 2.5.1 Experiment A: XGBoost Selected Variables

*The first experiment used only the variables selected through the XGBoost feature importance analysis. As described in the preprocessing stage, the XGBRegressor model identified the following Statspack metrics as the strongest predictors of runtime_minutes:*

- *plsql_body_pinhits*
- *enqueue_latch_gets*
- *cpu_call_time*
- *non_idle_waits*
- *dc_segments_gets*

*Because these variables were selected by a supervised model trained explicitly on runtime_minutes, the resulting clusters are expected to reflect clear differences in execution duration. This provides a useful perspective for assessing whether Datapatch executions naturally separate according to time. At the same time, two considerations must be acknowledged. First, the selection process introduces a bias toward time related separation, which may limit the discovery of other structural patterns. Second, although statistically relevant, the interpretability of these variables is constrained by the limited public documentation available for the underlying Statspack metrics.*

*For this experiment, the five XGBoost selected variables were standardized and evaluated across a full grid of clustering models using candidate solutions from two to five clusters. Silhouette Score served as the primary metric for assessing cluster quality, with the Calinski Harabasz and Davies Bouldin indexes used as secondary references. Principal Component Analysis was applied to visualize and interpret the resulting cluster structures.*

### 2.5.2 Experiment B: Semantic Engineered Variables

*The second experiment relied exclusively on the engineered variables created through the semantic strategy. These variables summarize higher level performance dimensions derived from Statspack counters and were constructed to capture the internal workload characteristics that may arise during Datapatch execution. The engineered variables included:*

- *Enqueue Pressure ($P_e$)*
- *I/O Pressure ($P_{io}$)*
- *CPU Pressure ($P_{cpu}$)*
- *Buffer Pressure ($P_b$)*
- *Parse Pressure ($P_p$)*

*These variables are highly interpretable because each one corresponds to a well defined subsystem of Oracle performance behavior, such as contention, disk access, CPU load, buffer cache efficiency or SQL parsing activity. Unlike the XGBoost selected variables, this feature set was not optimized for predicting elapsed time. As a result, the clusters obtained in this experiment are expected to reveal distinct behavioral patterns, but not necessarily large differences in execution duration. The emphasis of this experiment is interpretability rather than time driven separation.*

*As in Experiment A, the variables were standardized and each clustering model was evaluated over a grid of two to five clusters. Cluster quality was assessed with the same internal validation metrics, and PCA projections were generated to support qualitative interpretation of the resulting groups.*

### 2.5.3 Overall Experimental Procedure

*Both experiments followed the same methodological pipeline, structured as follows:*

1. *Select the variable set (XGBoost selected variables for Experiment A, semantic engineered variables for Experiment B).*
2. *Standardize all variables using* `StandardScaler` *to ensure comparable scales.*
3. *Apply each clustering model across candidate solutions containing two to five clusters.*
4. *Evaluate cluster quality using Silhouette Score as the primary metric and the Calinski Harabasz and Davies Bouldin indexes as secondary references.*
5. *Select the best performing configuration for each experiment based on the evaluation metrics.*
6. *Generate a two component Principal Component Analysis projection of the selected clustering solution.*
7. *Interpret the clusters using both the PCA visualization and the contribution of the variables to the principal components.*

# 3 Results and discussion

*Two types of experiments were conducted for this project. Experiment A generated clusters using the variables selected through the XGBoost model. Experiment B generated clusters using variables defined semantically on the basis of official Oracle documentation and was included in order to enhance interpretability. The results for each experiment are described in the following sections.*

## 3.1 Experiment A: Clusters with XGBoost

*The complete results and evaluations for this experiment are presented in Figure 3.1. The KMeans model with two clusters was excluded from consideration because the Spectral model with two clusters achieved a higher Silhouette validation score. The next configuration with the best performance and a different number of clusters, was again a Spectral model. For this reason the Spectral models with two and five clusters were selected for detailed examination.*

### 3.1.1 Spectral (k=2)

*The Spectral Clustering model with two clusters demonstrates a coherent and well defined structure according to the three evaluation metrics. A Silhouette score of 0.565356 indicates that observations are generally closer to their assigned cluster than to alternative ones, suggesting a moderate but reliable separation. The Calinski Harabasz index, which reached a value of 73.167484, reflects compact clusters with substantial dispersion between them, reinforcing the presence of a meaningful partition in the data. The Davies Bouldin index obtained a value of 0.450974, which is comparatively low and points to limited similarity between clusters and a high degree of internal cohesion.*

*Interpreted together, these metrics show that the two identified groups represent a stable structure, suggesting that the performance variables selected for the analysis form natural divisions that Spectral Clustering is able to detect with considerable clarity.*

| | model | k_proposed | k_found | silhouette | calinski_harabasz | davies_bouldin |
|---|---|---|---|---|---|---|
| **4** | Spectral | 2 | 2 | 0.565356 | 73.167484 | 0.450974 |
| **0** | KMeans | 2 | 2 | 0.544192 | 196.870707 | 0.775374 |
| **7** | Spectral | 5 | 5 | 0.507266 | 83.033856 | 0.484802 |
| **19** | MeanShift | 5 | 4 | 0.500681 | 127.151563 | 0.808808 |
| **16** | MeanShift | 2 | 4 | 0.500681 | 127.151563 | 0.808808 |
| **17** | MeanShift | 3 | 4 | 0.500681 | 127.151563 | 0.808808 |
| **18** | MeanShift | 4 | 4 | 0.500681 | 127.151563 | 0.808808 |
| **13** | GaussianMixture | 3 | 3 | 0.497476 | 130.239470 | 0.792161 |
| **1** | KMeans | 3 | 3 | 0.495660 | 147.134222 | 0.796988 |
| **9** | Ward | 3 | 3 | 0.477755 | 177.715549 | 0.797673 |
| **8** | Ward | 2 | 2 | 0.476373 | 175.580224 | 0.809664 |
| **6** | Spectral | 4 | 4 | 0.472967 | 74.676289 | 0.541926 |
| **10** | Ward | 4 | 4 | 0.471906 | 170.856571 | 0.830217 |
| **3** | KMeans | 5 | 5 | 0.460075 | 188.660292 | 0.813212 |
| **15** | GaussianMixture | 5 | 5 | 0.437648 | 175.034179 | 0.844728 |
| **11** | Ward | 5 | 5 | 0.437133 | 179.744910 | 0.921632 |
| **2** | KMeans | 4 | 4 | 0.419791 | 175.355517 | 0.939402 |
| **5** | Spectral | 3 | 3 | 0.400352 | 47.255701 | 0.476337 |
| **14** | GaussianMixture | 4 | 4 | 0.377358 | 158.288592 | 0.942184 |
| **12** | GaussianMixture | 2 | 2 | 0.307430 | 89.597789 | 1.075468 |
| **20** | DBSCAN | 2 | 4 | 0.131360 | 23.019581 | 0.995244 |
| **21** | DBSCAN | 3 | 4 | 0.131360 | 23.019581 | 0.995244 |
| **22** | DBSCAN | 4 | 4 | 0.131360 | 23.019581 | 0.995244 |
| **23** | DBSCAN | 5 | 4 | 0.131360 | 23.019581 | 0.995244 |

Figure 3.1: Results of the grid for clustering models.

*The resulting PCA projection using two components for Spectral (k=2) is represented in Figure 3.2*

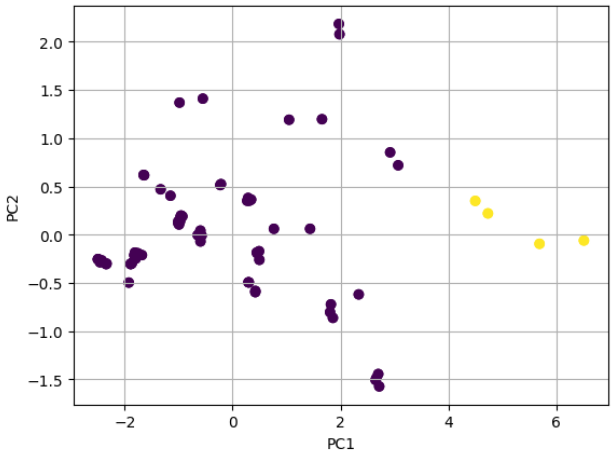


Figure 3.2: Spectral clusters PCA with 2 clusters

**Cluster 0**

*Cluster 0 contained 136 of the 144 observations. When examining the runtime_minutes variable, this cluster presented a mean value of 14.48 minutes. Cluster 0 represents the typical and most stable execution pattern observed during Datapatch. The values of plsql_body_pinhits indicate a regular and balanced use of PL/SQL objects, suggesting that the database is able to retrieve and reuse these structures efficiently without excessive recompilation. In parallel, the metric enqueue_latch_gets remains at moderate levels, which implies that contention for enqueue related latches is limited and that metadata operations proceed without significant serialization. This stable behavior extends to cpu_call_time, where the observed values reflect a reasonable consumption of CPU resources per database call, characteristic of an environment operating without substantial processing demands.*

*The remaining indicators align with this interpretation. The number of non_idle_waits is present but not excessive, suggesting that the system encounters resource waits at frequencies typical of routine activity rather than sustained bottlenecks. Similarly, dc_segments_gets shows that dictionary segment lookups occur at a normal rate. As a consequence of this overall balance and low contention, the runtime_minutes for Datapatch runs in Cluster 0 remains comparatively low, reinforcing the view that this cluster reflects efficient and predictable system behavior.*

*Description of cluster 0 can be found in Figure 3.3 and a heatmap representation can be observed in Figure 3.4*

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| plsql_body_pinhits | 136.0 | 25874.205882 | 18895.357822 | 4650.000000 | 7338.50 | 26454.500000 | 4.243275e+04 | 6.695800e+04 |
| enqueue_latch_gets | 136.0 | 747880.647059 | 522316.460882 | 214892.000000 | 297214.50 | 579792.000000 | 1.078151e+06 | 2.166649e+06 |
| cpu_call_time | 136.0 | 64483.382353 | 45030.693482 | 11302.000000 | 39090.50 | 51413.000000 | 8.574800e+04 | 1.672400e+05 |
| non_idle_waits | 136.0 | 93859.632353 | 75474.791762 | 5129.000000 | 42003.25 | 84487.500000 | 1.320832e+05 | 3.588620e+05 |
| dc_segments_gets | 136.0 | 16057.838235 | 9502.876551 | 1723.000000 | 10366.00 | 15240.000000 | 2.097875e+04 | 4.231800e+04 |
| runtime_minutes | 136.0 | 14.484514 | 8.379282 | 0.777256 | 9.04 | 11.891882 | 1.929750e+01 | 3.863853e+01 |
| cluster | 136.0 | 0.000000 | 0.000000 | 0.000000 | 0.00 | 0.000000 | 0.000000e+00 | 0.000000e+00 |

Figure 3.3: Spectral (k=2) description of cluster 0

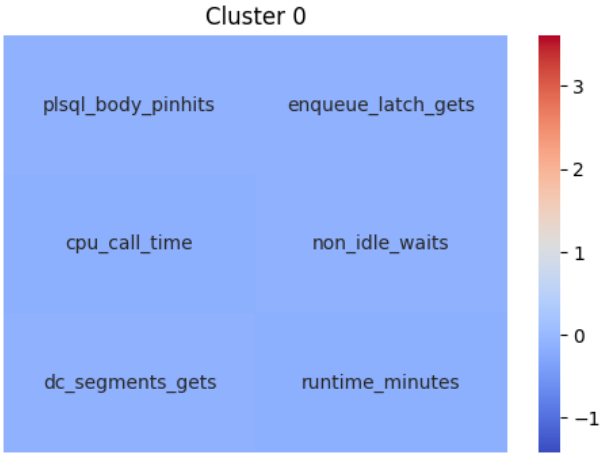


Figure 3.4: Spectral (k=2) heatmap representation of cluster 0

**Cluster 1**

*Cluster 1 contained only 8 observations. These samples showed a mean runtime_minutes value of 45.28 minutes, which is significantly higher than the baseline cluster. Cluster 1 displays a distinctly elevated performance profile, pointing to an execution mode characterized by substantially heavier internal activity. The sharp increase in plsql_body_pinhits suggests a higher frequency of PL/SQL object access, which may stem from repeated execution or recompilation of PL/SQL routines triggered by more complex patch operations. This behavior is accompanied by a significant rise in enqueue_latch_gets, revealing pronounced contention around enqueue latches. Such contention is typically associated with operations that require serialized access to metadata or structural components of the database, indicating that Datapatch is performing more intensive or more frequent DDL like actions during these executions.*

*A similar pattern appears in the CPU and waiting time metrics. The variable cpu_call_time increases markedly, demonstrating that database calls consume substantially more CPU resources under this cluster, likely as a result of recursive SQL or dictionary lookups. The notable elevation of non_idle_waits further reinforces this interpretation, showing that the system repeatedly encounters conditions that force it to pause and wait for resources, locks, or*

*I/O before progressing. Additionally, dc_segments_gets rises significantly, confirming that dictionary segment metadata is accessed at a much higher rate, consistent with patch operations that modify numerous schema objects.*

*This combination of elevated internal activity and sustained contention leads to a substantial increase in the elapsed time associated with Cluster 1. The metrics collectively suggest that Datapatch executions in this cluster experience a high load environment marked by contention, heavy dictionary interaction, and increased CPU usage, resulting in prolonged completion times.*

*Description of cluster 0 can be found in Figure 3.5 and a heatmap representation can be observed in Figure 3.6*

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| plsql_body_pinhits | 8.0 | 7.679750e+04 | 12107.671842 | 6.399000e+04 | 6.653700e+04 | 7.634100e+04 | 8.660150e+04 | 90518.00 |
| enqueue_latch_gets | 8.0 | 2.319029e+06 | 456443.397017 | 1.855489e+06 | 1.948894e+06 | 2.251824e+06 | 2.621960e+06 | 2916979.00 |
| cpu_call_time | 8.0 | 2.259465e+05 | 10878.546134 | 2.136010e+05 | 2.184618e+05 | 2.247920e+05 | 2.322768e+05 | 240601.00 |
| non_idle_waits | 8.0 | 3.210962e+05 | 32388.298296 | 2.910820e+05 | 2.924838e+05 | 3.155715e+05 | 3.441840e+05 | 362160.00 |
| dc_segments_gets | 8.0 | 4.465900e+04 | 2182.570307 | 4.194500e+04 | 4.374875e+04 | 4.449850e+04 | 4.540875e+04 | 47694.00 |
| runtime_minutes | 8.0 | 4.528420e+01 | 6.380450 | 3.994598e+01 | 4.002049e+01 | 4.384104e+01 | 4.693873e+01 | 55.35 |
| cluster | 8.0 | 1.000000e+00 | 0.000000 | 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 1.00 |

Figure 3.5: Spectral (k=2) description of cluster 1

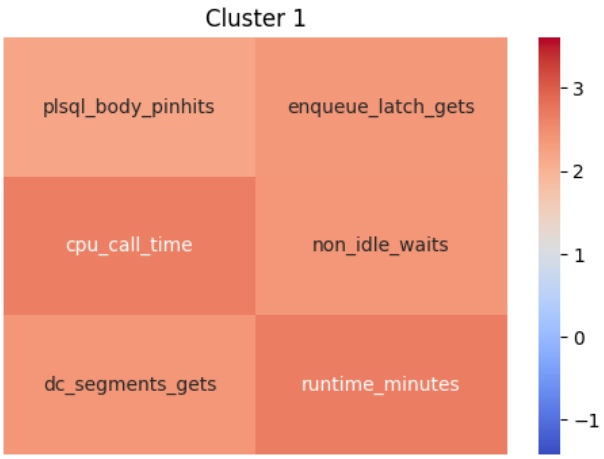


Figure 3.6: Spectral (k=2) heatmap representation of cluster 1

**Summary**

*Cluster 0 represents efficient, low contention Datapatch executions, whereas Cluster 1 captures a high intensity operational mode characterized by contention, elevated CPU usage, heavy dictionary demand, and significantly longer runtimes.*

### 3.1.2 *Spectral (k=5)*

*The five cluster configuration provided the next most coherent structure. This model preserved the separation observed in the two cluster configuration while offering a more granular view of the data.*

*The resulting PCA projection using two components for Spectral (k=5) is represented in Figure 3.7*

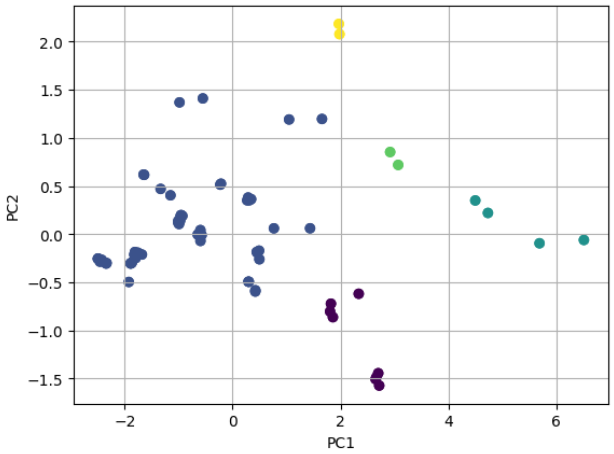


Figure 3.7: Spectral clusters PCA with 5 clusters

**Cluster 0**

*Cluster 0 reflects a balanced and steady execution mode in which Datapatch exhibits moderate activity across all variables. The levels of plsql_body_pinhits and dc_segments_gets indicate routine access to PL/SQL objects and dictionary metadata without excessive reloads or lookups, while enqueue_latch_gets and non_idle_waits remain controlled and do not suggest meaningful contention. cpu_call_time stays at a moderate level, and the corresponding runtime_minutes is reasonably short, positioning this cluster as a normal operational state where the system works consistently but without signs of internal stress.*

*Description of cluster 0 can be found in Figure 3.8 and a heatmap representation can be observed in Figure 3.9*

**Cluster 1**

*Cluster 1 represents the lightest workload profile, characterized by minimal internal activity across all variables. The very low values of plsql_body_pinhits, enqueue_latch_gets, and dc_segments_gets indicate that PL/SQL operations, locking structures, and dictionary lookups are barely exercised. Likewise,*

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| plsql_body_pinhits | 16.0 | 6.039025e+04 | 5848.698197 | 5.318800e+04 | 5.503750e+04 | 61442.0 | 6.556550e+04 | 6.695800e+04 |
| enqueue_latch_gets | 16.0 | 1.798527e+06 | 372948.379894 | 1.376006e+06 | 1.424038e+06 | 1856831.0 | 2.153052e+06 | 2.166649e+06 |
| cpu_call_time | 16.0 | 1.516581e+05 | 11098.874795 | 1.345690e+05 | 1.427070e+05 | 153403.5 | 1.607438e+05 | 1.654260e+05 |
| non_idle_waits | 16.0 | 1.205556e+05 | 11638.810133 | 1.076820e+05 | 1.081888e+05 | 121135.0 | 1.307242e+05 | 1.358930e+05 |
| dc_segments_gets | 16.0 | 2.436525e+04 | 4469.428651 | 2.018000e+04 | 2.081375e+04 | 23348.5 | 2.628100e+04 | 3.348000e+04 |
| runtime_minutes | 16.0 | 2.880611e+01 | 2.106007 | 2.529559e+01 | 2.729738e+01 | 29.5 | 3.056094e+01 | 3.173620e+01 |
| cluster | 16.0 | 0.000000e+00 | 0.000000 | 0.000000e+00 | 0.000000e+00 | 0.0 | 0.000000e+00 | 0.000000e+00 |

Figure 3.8: Spectral (k=5) description of cluster 0

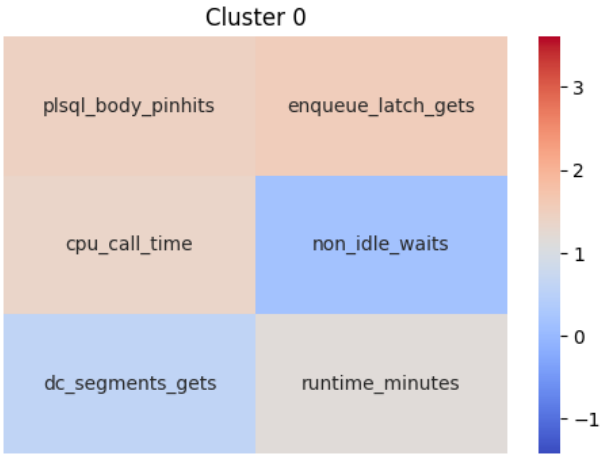


Figure 3.9: Spectral (k=5) heatmap representation of cluster 0

*cpu_call_time and non_idle_waits remain minimal, showing that processing and waiting demands are extremely low. As a result, runtime_minutes is the shortest of all clusters, making Cluster 1 the clearest expression of an optimal, low overhead Datapatch execution with virtually no performance pressure.*

*Description of cluster 1 can be found in Figure 3.10 and a heatmap representation can be observed in Figure 3.11*

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| plsql_body_pinhits | 112.0 | 19656.446429 | 13981.266569 | 4650.000000 | 6506.750000 | 11310.500000 | 32161.750000 | 4.881100e+04 |
| enqueue_latch_gets | 112.0 | 564335.321429 | 320712.917171 | 214892.000000 | 296001.000000 | 477178.000000 | 818488.500000 | 1.215033e+06 |
| cpu_call_time | 112.0 | 48800.428571 | 27372.573515 | 11302.000000 | 31585.750000 | 48586.000000 | 63656.000000 | 1.075840e+05 |
| non_idle_waits | 112.0 | 75461.660714 | 56888.457324 | 5129.000000 | 13449.250000 | 78650.000000 | 93007.250000 | 2.133340e+05 |
| dc_segments_gets | 112.0 | 13463.303571 | 7704.993200 | 1723.000000 | 8494.750000 | 13476.000000 | 18886.000000 | 3.642300e+04 |
| runtime_minutes | 112.0 | 11.778725 | 5.854243 | 0.777256 | 8.711721 | 10.636336 | 15.970082 | 2.679731e+01 |
| cluster | 112.0 | 1.000000 | 0.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000e+00 |

Figure 3.10: Spectral (k=5) description of cluster 1

### Cluster 2

*Cluster 2 is the most resource intensive and demanding execution pattern in the model. Extremely high values of plsql_body_pinhits, enqueue_latch_gets,*

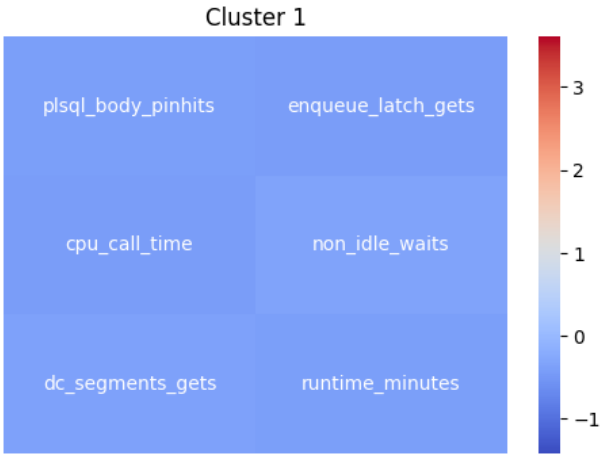


Figure 3.11: Spectral (k=5) heatmap representation of cluster 1

*dc_segments_gets, cpu_call_time, and non_idle_waits reveal heavy PL/SQL usage, significant dictionary interaction, and pronounced latch contention. These combined pressures produce the longest runtime_minutes, indicating that Datapatch is operating under sustained internal stress and frequent bottlenecks. Cluster 2 therefore captures the system at its peak workload intensity, where complex operations and synchronization requirements dominate performance.*

*Description of cluster 2 can be found in Figure 3.12 and a heatmap representation can be observed in Figure 3.13*

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| plsql_body_pinhits | 8.0 | 7.679750e+04 | 12107.671842 | 6.399000e+04 | 6.653700e+04 | 7.634100e+04 | 8.660150e+04 | 90518.00 |
| enqueue_latch_gets | 8.0 | 2.319029e+06 | 456443.397017 | 1.855489e+06 | 1.948894e+06 | 2.251824e+06 | 2.621960e+06 | 2916979.00 |
| cpu_call_time | 8.0 | 2.259465e+05 | 10878.546134 | 2.136010e+05 | 2.184618e+05 | 2.247920e+05 | 2.322768e+05 | 240601.00 |
| non_idle_waits | 8.0 | 3.210962e+05 | 32388.298296 | 2.910820e+05 | 2.924838e+05 | 3.155715e+05 | 3.441840e+05 | 362160.00 |
| dc_segments_gets | 8.0 | 4.465900e+04 | 2182.570307 | 4.194500e+04 | 4.374875e+04 | 4.449850e+04 | 4.540875e+04 | 47694.00 |
| runtime_minutes | 8.0 | 4.528420e+01 | 6.380450 | 3.994598e+01 | 4.002049e+01 | 4.384104e+01 | 4.693873e+01 | 55.35 |
| cluster | 8.0 | 2.000000e+00 | 0.000000 | 2.000000e+00 | 2.000000e+00 | 2.000000e+00 | 2.000000e+00 | 2.00 |

Figure 3.12: Spectral (k=5) description of cluster 2

### Cluster 3

*Cluster 3 reflects a moderately high workload state in which multiple variables exceed baseline levels but remain below the extreme values observed in Cluster 2. Elevated plsql_body_pinhits, enqueue_latch_gets, and dc_segments_gets suggest heavier than usual PL/SQL and dictionary activity, while increases in cpu_call_time and non_idle_waits indicate heightened processing demands and periodic waiting. runtime_minutes is longer than in low load clusters but still manageable, positioning Cluster 3 as a heavier yet*

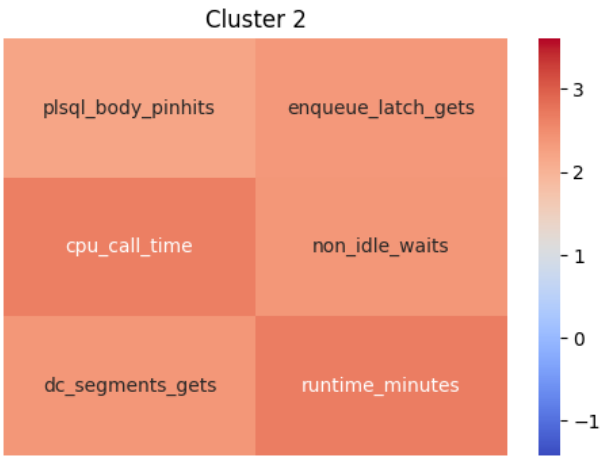


Figure 3.13: Spectral (k=5) heatmap representation of cluster 2

*controlled execution scenario that reflects a more complex internal workload without critical contention.*

*Description of cluster 3 can be found in Figure 3.14 and a heatmap representation can be observed in Figure 3.15*

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| plsql_body_pinhits | 4.0 | 4.405400e+04 | 110.851252 | 4.395800e+04 | 4.395800e+04 | 44054.0 | 4.415000e+04 | 4.415000e+04 |
| enqueue_latch_gets | 4.0 | 1.336993e+06 | 71643.394904 | 1.274948e+06 | 1.274948e+06 | 1336993.0 | 1.399038e+06 | 1.399038e+06 |
| cpu_call_time | 4.0 | 1.635315e+05 | 4282.206947 | 1.598230e+05 | 1.598230e+05 | 163531.5 | 1.672400e+05 | 1.672400e+05 |
| non_idle_waits | 4.0 | 2.423365e+05 | 1180.681300 | 2.413140e+05 | 2.413140e+05 | 242336.5 | 2.433590e+05 | 2.433590e+05 |
| dc_segments_gets | 4.0 | 4.224800e+04 | 80.829038 | 4.217800e+04 | 4.217800e+04 | 42248.0 | 4.231800e+04 | 4.231800e+04 |
| runtime_minutes | 4.0 | 3.352590e+01 | 4.084959 | 3.006506e+01 | 3.031626e+01 | 32.7 | 3.590963e+01 | 3.863853e+01 |
| cluster | 4.0 | 3.000000e+00 | 0.000000 | 3.000000e+00 | 3.000000e+00 | 3.0 | 3.000000e+00 | 3.000000e+00 |

Figure 3.14: Spectral (k=5) description of cluster 3

### Cluster 4

*Cluster 4 shows a low to moderate workload profile characterized by consistent but modest values across the variables. Metrics such as plsql_body_pinhits, enqueue_latch_gets, dc_segments_gets, and non_idle_waits remain limited, indicating a lightly loaded system with minimal contention and predictable metadata access patterns. cpu_call_time stays low as well, supporting short and stable runtime_minutes. This cluster represents an efficient execution state that involves slightly more internal work than the extremely light Cluster 1 but remains far from any contention heavy conditions.*

*Description of cluster 4 can be found in Figure 3.16 and a heatmap representation can be observed in Figure 3.17*

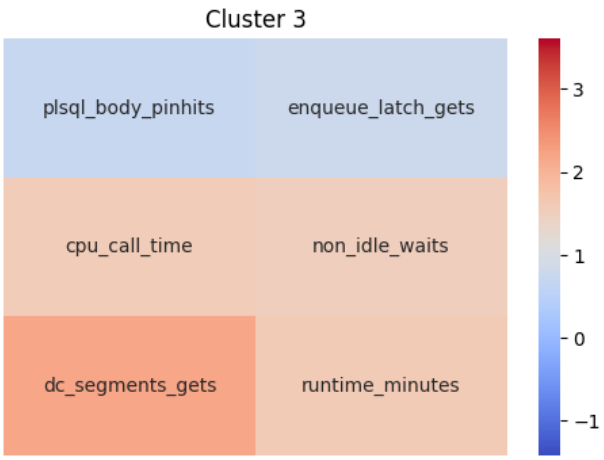


Figure 3.15: Spectral (k=5) heatmap representation of cluster 3

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| plsql_body_pinhits | 4.0 | 4.372750e+04 | 92.953393 | 43647.00 | 4.364700e+04 | 4.372750e+04 | 4.380800e+04 | 43808.00 |
| enqueue_latch_gets | 4.0 | 1.095454e+06 | 39957.834780 | 1060849.00 | 1.060849e+06 | 1.095454e+06 | 1.130058e+06 | 1130058.00 |
| cpu_call_time | 4.0 | 5.585900e+04 | 160.503375 | 55720.00 | 5.572000e+04 | 5.585900e+04 | 5.599800e+04 | 55998.00 |
| non_idle_waits | 4.0 | 3.537420e+05 | 5912.066757 | 348622.00 | 3.486220e+05 | 3.537420e+05 | 3.588620e+05 | 358862.00 |
| dc_segments_gets | 4.0 | 2.928500e+04 | 147.801669 | 29157.00 | 2.915700e+04 | 2.928500e+04 | 2.941300e+04 | 29413.00 |
| runtime_minutes | 4.0 | 1.391884e+01 | 1.235379 | 12.65 | 1.297243e+01 | 1.395268e+01 | 1.489909e+01 | 15.12 |
| cluster | 4.0 | 4.000000e+00 | 0.000000 | 4.00 | 4.000000e+00 | 4.000000e+00 | 4.000000e+00 | 4.00 |

Figure 3.16: Spectral (k=5) description of cluster 4

**Summary**

*Each cluster highlights a distinct Datapatch execution, ranging from optimal and minimal load behavior (Cluster 1), through efficient but moderately active patterns (Clusters 4 and 0), to increasingly resource demanding and complex execution states (Clusters 3 and 2).*

## 3.2 Experiment B: Clusters with Semantic Variables

*This experiment used variables selected according to Oracle documentation for Statspack[5]. The main objective was to generate clusters that were easier to interpret from a DBA perspective, since these variables have clearer operational meaning compared to the internal Oracle metrics used in the previous experiment.*

*Only the top performing clustering configurations were considered. The ranking of model performance and the scoring of each configuration are shown in Figure 3.18. This selection approach ensured that only meaningful and well separated partitions were reviewed in the following analysis.*

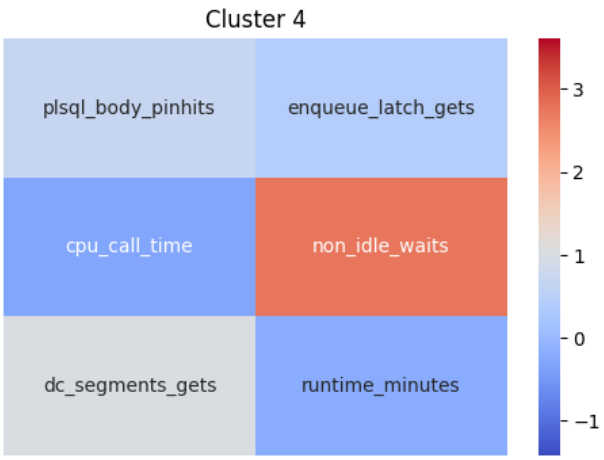


Figure 3.17: Spectral (k=5) heatmap representation of cluster 4

| | modelo | k_propuesto | clusters_encontrados | silhouette | calinski_harabasz | davies_bouldin |
|---|---|---|---|---|---|---|
| 3 | Spectral | 3 | 3 | 0.531738 | 19.712754 | 0.325170 |
| 4 | Spectral | 4 | 4 | 0.493592 | 18.581264 | 0.332626 |
| 14 | MeanShift | 5 | 5 | 0.426994 | 25.557922 | 0.644064 |
| 13 | MeanShift | 4 | 5 | 0.426994 | 25.557922 | 0.644064 |
| 12 | MeanShift | 3 | 5 | 0.426994 | 25.557922 | 0.644064 |
| 9 | GaussianMixture | 3 | 3 | 0.368617 | 48.103711 | 1.223811 |

Figure 3.18: Results of the grid search using semantic variables.

*As shown in the figure 3.18, the best performing configuration was the Spectral model with three clusters. This configuration achieved the strongest internal validation metrics and produced a meaningful separation of the data. Therefore, it was selected as the final model for interpretation in this experiment.*

### 3.2.1 Spectral (k=3)

*The Spectral model with three clusters demonstrated a clear and interpretable structure. The Silhouette score of 0.531738 reflects a reasonable separation between clusters, with observations remaining closer to their assigned group than to others. The Calinski Harabasz index reached 19.712754, indicating compact clusters with acceptable dispersion. The Davies Bouldin index obtained a value of 0.325170, which suggests limited overlap and consistent internal cohesion.*

*Overall, these values indicate that the three clusters form a meaningful and stable configuration. This structure provides a finer level of detail than the two cluster model and reveals distinct execution patterns that are relevant for interpreting Datapatch performance under the selected semantic variables.*

*The PCA projection shown in Figure 3.19 supports this interpretation. Most observations form a dense central group, representing the predominant and stable execution pattern. Two smaller groups appear separated from this region, showing noticeably different behavior. One group is positioned to the far left of the projection and the second appears slightly above the central mass, each occupying isolated regions with no overlap.*

*This spatial separation suggests that these groups reflect specific execution behaviors not observed in the main cluster. In practical terms, the central group represents normal and consistent runs, while the isolated points likely correspond to executions influenced by higher pressure events, resource contention or external conditions. The PCA visualization therefore reinforces the validity of the three cluster structure and confirms the distinctions identified by the semantic variables.*

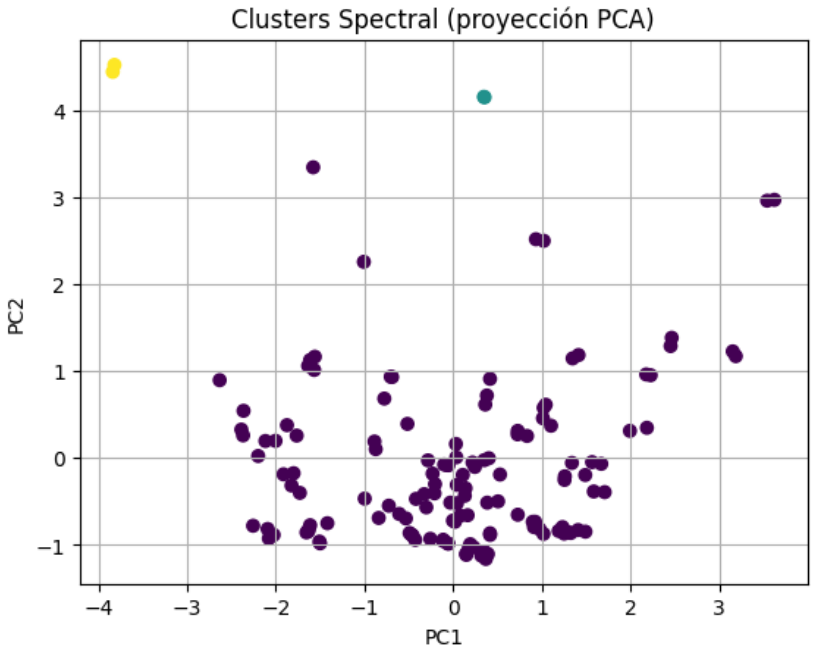


Figure 3.19: Clusters obtained using semantic variables. PCA projection.

**Cluster 0**

*Cluster 0 contains the majority of executions and represents a stable performance pattern. The mean value of runtime_minutes in this group is 16.19 minutes, indicating that most executions completed within a consistent and predictable duration. The remaining variables show low values across enqueue, I/O, CPU, buffer and parse pressure, suggesting limited contention and mini-*

*mal resource impact during execution.*

*The distribution of values supports this interpretation. Median values remain low and upper quartiles show only modest increases, which indicates that occasional fluctuations do not escalate to levels that affect runtime in a meaningful way. The behavior observed in this group aligns with the expected performance of Datapatch under normal workload conditions.*

*From a practical perspective, this cluster reflects healthy and expected execution behavior. It therefore serves as the reference baseline against which the remaining clusters can be compared.*

*A detailed summary of this cluster is presented in Figure 3.20, and a heatmap representation can be observed in Figure 3.21.*

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| enqueue_pressure | 140.0 | 9.089776 | 9.038088 | 1.710098 | 3.794732 | 5.973702 | 12.369883 | 62.934363 |
| redo_pressure | 140.0 | 1.321141 | 1.749905 | 0.000000 | 0.000000 | 0.501916 | 2.000000 | 8.000000 |
| io_pressure | 140.0 | 1.428962 | 1.608865 | 0.000000 | 0.179894 | 0.682431 | 2.258740 | 7.507439 |
| cpu_pressure | 140.0 | 0.369504 | 0.164065 | 0.025659 | 0.252892 | 0.351939 | 0.452641 | 0.895810 |
| buffer_pressure | 140.0 | 1.816774 | 2.525565 | 0.010000 | 0.289800 | 0.488053 | 2.273454 | 12.304152 |
| parse_pressure | 140.0 | 0.346676 | 0.144891 | 0.057481 | 0.284478 | 0.378155 | 0.462512 | 0.585614 |
| elapsed_minutes | 140.0 | 16.198404 | 11.033790 | 0.777256 | 9.167500 | 12.110430 | 21.604753 | 55.350000 |

Figure 3.20: Spectral (k=3) statistical summary of cluster 0.

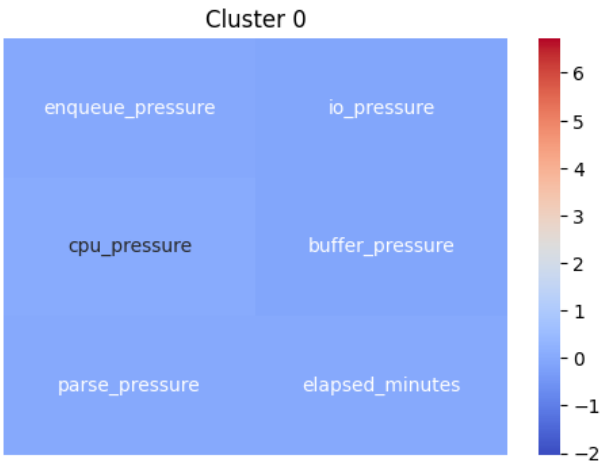


Figure 3.21: Spectral (k=3) heatmap representation of cluster 0.

### Cluster 1

*Cluster 1 contains two executions that display a distinct behavior when*

*compared to the baseline. The mean value of runtime_minutes in this group is 14.97 minutes, which remains close to the runtime observed in Cluster 0. However, the pressure metrics are substantially higher, particularly in enqueue, I/O and buffer activity. These elevated values indicate that the executions in this cluster operated under noticeably higher internal demand.*

*Despite the increased pressure, the elapsed time did not rise significantly. This suggests that the system was able to handle these bursts of activity without performance degradation. The low variation observed across the metrics in this group indicates that the behavior is consistent and not caused by random fluctuation.*

*In practical terms, this cluster represents executions where internal pressure increased, yet performance remained stable. These runs may correspond to short-lived contention, increased data movement or momentary load conditions that did not translate into longer execution times. As a result, Cluster 1 reflects an execution pattern that differs from the baseline while remaining efficient.*

*A detailed summary for this cluster is shown in Figure 3.22, and a heatmap representation is provided in Figure 3.23.*

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| enqueue_pressure | 2.0 | 21.507853 | 0.299187 | 21.296296 | 21.402075 | 21.507853 | 21.613632 | 21.719411 |
| redo_pressure | 2.0 | 4.846154 | 0.000000 | 4.846154 | 4.846154 | 4.846154 | 4.846154 | 4.846154 |
| io_pressure | 2.0 | 3.917755 | 0.030790 | 3.895983 | 3.906869 | 3.917755 | 3.928641 | 3.939527 |
| cpu_pressure | 2.0 | 0.235871 | 0.003224 | 0.233591 | 0.234731 | 0.235871 | 0.237010 | 0.238150 |
| buffer_pressure | 2.0 | 16.326654 | 0.033016 | 16.303308 | 16.314981 | 16.326654 | 16.338327 | 16.350000 |
| parse_pressure | 2.0 | 0.409098 | 0.005125 | 0.405475 | 0.407286 | 0.409098 | 0.410910 | 0.412722 |
| elapsed_minutes | 2.0 | 14.972724 | 0.208280 | 14.825448 | 14.899086 | 14.972724 | 15.046362 | 15.120000 |

Figure 3.22: Spectral (k=3) statistical summary of cluster 1.

### Cluster 2

*Cluster 2 contains two executions that exhibit the highest levels of resource activity among the three groups. The most distinctive characteristic of this cluster is the elevated _pressure value, which reaches a mean of 19.52 and is considerably higher than the values observed in the previous clusters. Enqueue and buffer pressure also present noticeable increases, indicating that these executions encountered both data movement demand and internal coordination activity during runtime.*

*The mean value of runtime_minutes for this group is 17.22 minutes, which*

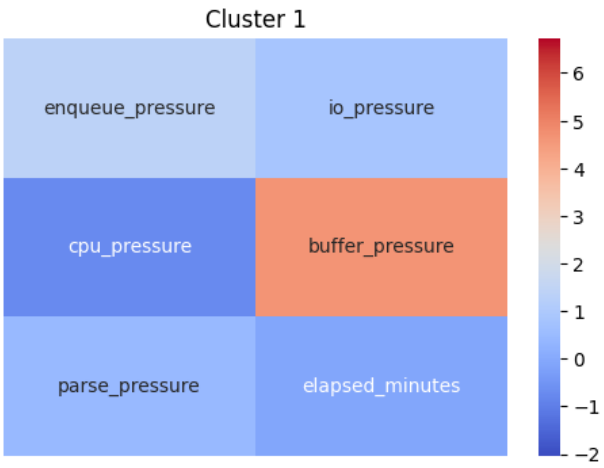


Figure 3.23: Spectral (k=3) heatmap representation of cluster 1.

*is slightly higher than the baseline observed in Cluster 0 and in a similar range to Cluster 1. This suggests that although Cluster 2 experienced substantial activity, particularly at the I/O level, the execution time remained within an acceptable range. CPU and parse pressure remain low, indicating that the workload did not escalate into full resource contention across subsystems.*

*In practical terms, Cluster 2 reflects executions where I/O pressure dominated the runtime signature. These runs likely correspond to conditions where data access requirements increased, possibly due to larger metadata operations or increased read and write activity, without triggering a proportional increase in overall execution duration. This cluster therefore represents a distinct performance state driven by I/O intensity rather than broad system contention.*

*A detailed summary for this cluster is shown in Figure 3.24, and a heatmap representation of its variables is provided in Figure 3.25.*

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **enqueue_pressure** | 2.0 | 12.145988 | 1.293845 | 11.231102 | 11.688545 | 12.145988 | 12.603431 | 13.060874 |
| **io_pressure** | 2.0 | 19.524461 | 0.079558 | 19.468204 | 19.496333 | 19.524461 | 19.552589 | 19.580717 |
| **cpu_pressure** | 2.0 | 0.070496 | 0.007925 | 0.064892 | 0.067694 | 0.070496 | 0.073298 | 0.076099 |
| **buffer_pressure** | 2.0 | 6.577732 | 0.010934 | 6.570000 | 6.573866 | 6.577732 | 6.581597 | 6.585463 |
| **parse_pressure** | 2.0 | 0.155441 | 0.001539 | 0.154353 | 0.154897 | 0.155441 | 0.155985 | 0.156529 |
| **elapsed_minutes** | 2.0 | 17.222714 | 1.834640 | 15.925427 | 16.574071 | 17.222714 | 17.871357 | 18.520000 |

Figure 3.24: Spectral (k=3) statistical summary of cluster 2.

### Summary

*In summary, the three clusters reveal distinct execution patterns under*

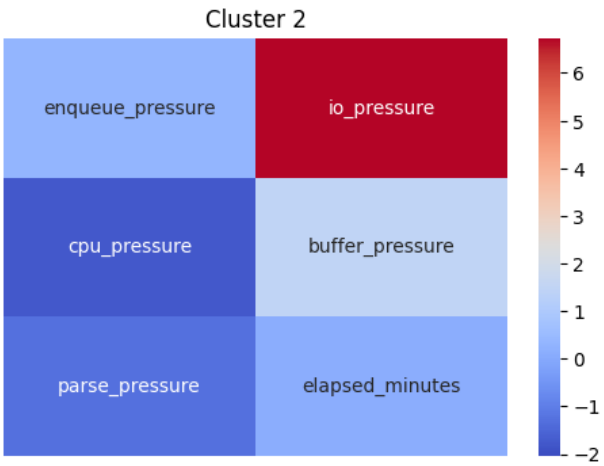


Figure 3.25: Spectral (k=3) heatmap representation of cluster 2.

*the selected semantic variables. Cluster 0 represents the baseline behavior and reflects consistent runs with minimal pressure across all metrics. Cluster 1 shows increased internal activity while maintaining execution times similar to the baseline, suggesting short lived or localized resource demand that the system handled efficiently. Cluster 2 displays the highest levels of I/O related pressure, although execution time remained within an acceptable range. Together, these results indicate that Datapatch can operate under different degrees of load without experiencing severe degradation, and that the semantic variables provide a meaningful lens to distinguish these execution states.*

# 4 Conclusions and Future Work

## 4.1 Conclusions

*This work presented the first structured attempt to characterize Datapatch execution behavior using performance telemetry collected during live patching. Since no dataset or reference framework previously existed, the initial challenge was to create a reliable data acquisition process capable of consistently capturing runtime behavior. This led to the development of an automated pipeline based on Statspack, chosen for its native availability, license free nature and standardized metric structure.*

*Once the dataset was established, a feature selection strategy was required to identify relevant performance signals. Two complementary approaches were applied. The first relied on a supervised model to extract statistically influential variables associated with execution duration. The second was based on semantic interpretation derived from Oracle documentation, prioritizing interpretability and operational meaning. This dual approach addressed one of the key objectives of the project by balancing analytical rigor with domain relevance.*

*With the selected variables, multiple clustering models were evaluated in order to identify recurring behavioral patterns during live patching. Spectral Clustering consistently produced the most coherent structures. In Experiment A, the model using statistically selected variables resulted in clusters strongly aligned with elapsed time, enabling a clear distinction between normal executions and potential anomalies. This outcome demonstrates that elapsed time reflects meaningful underlying system behavior and that clustering can serve as a reliable method for identifying outlier executions.*

*Experiment B introduced a semantic variable set and enabled a complementary interpretation. The resulting three cluster model showed that Datapatch can operate under different resource conditions without experiencing severe performance degradation. The clusters reflected three meaningful execution states: stable runs with low pressure, runs with transient internal spikes handled efficiently and runs dominated by I/O demand. Importantly,*

*these states were interpretable in operational terms, satisfying the objective of validating cluster meaning against real database behavior.*

*Overall, the findings confirm that unsupervised learning can be applied to characterize Datapatch behavior and that both statistical and semantic representations provide value. The results address the general objective by demonstrating that Oracle RDBMS runtime behavior during patching exhibits identifiable and recurring patterns that can be modeled, interpreted and compared. The project also met the specific objectives by establishing a data collection framework, defining a reproducible feature selection process, evaluating multiple clustering models and ensuring that interpretation remained grounded in operational relevance.*

*Finally, an important conclusion emerges from the process: analytical techniques alone are insufficient. The interpretability and usefulness of the findings depend on the ability to contextualize them within Oracle RDBMS architecture and runtime behavior. Clustering provides structure, but domain knowledge gives meaning. This research therefore demonstrates not only a technical methodology, but also a framework for reasoning about live patching performance using both data and expertise.*

## 4.2 Future Work

*Future research can build upon the results presented in this project in several meaningful directions. One of the most relevant opportunities is expanding the dataset. Although the current results demonstrate clear behavioral structure, the number of environments and execution types remains limited when compared to the heterogeneity of real world Oracle deployments. Databases vary significantly in workload profile, hardware configuration, storage technology, patch cadence, memory architecture and user concurrency. A larger and more diverse dataset, ideally sourced from production grade environments, would provide a stronger empirical basis and improve the generalizability of the discovered behavioral patterns.*

*Another direction relates to the interpretability of internal metrics. While clustering successfully identified execution patterns, the full operational meaning of these patterns requires a deeper understanding of Oracle RDBMS internals. Concepts such as buffer cache mechanics, latch contention, I/O scheduling, session concurrency, redo generation behavior and adaptive runtime decisions must be incorporated to refine interpretation. Statistical patterns reveal correlations, but a richer diagnostic model requires mapping these observations to concrete architectural mechanisms and known system*

*behaviors.*

*Future work may therefore explore the integration of additional Oracle instrumentation sources, such as Active Session History, wait events or AWR derived views when available. Combining Statspack with other sources could help bridge the gap between observed performance signals and the specific subsystems responsible for them. Such integration may also enable the development of more precise decision frameworks capable of linking cluster membership with actionable tuning or configuration recommendations.*

*A final area of exploration concerns automation and operational adoption. Once validated with a broader dataset, the methodology developed in this research could evolve into a monitoring or advisory component integrated into patching workflows. Such a tool could detect performance anomalies, compare current behavior against historical baselines and offer guidance based on recognized execution patterns. This would move the work from experimental characterization toward practical diagnostic assistance, contributing to higher reliability and predictability of live patching operations in production environments.*

# *Bibliography*


[1] Oracle Base, "Automatic workload repository (awr) in oracle 10g." https://oracle-base.com/articles/10g/automatic-workload-repository-10g, 2004.

[2] Oracle Corporation, "Statspack appendix." https://docs.oracle.com/cd/E13160_01/wli/docs10gr3/dbtuning/statsApdx.html, 2008.

[3] M. Zahn, "Oracle statspack survival guide." https://www.akadia.com/services/ora_statspack_survival_guide.html, 2005.

[4] XGBoost Developers, "Xgboost documentation: Python api reference (xgbregressor)." https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBRegressor, 2024.

[5] Oracle, "Top wli database bottlenecks." https://docs.oracle.com/cd/E13160_01/wli/docs10gr3/dbtuning/statsApdx.html.

[6] scikit-learn developers, "K-means clustering — scikit-learn 1.7.2 documentation." https://scikit-learn.org/stable/modules/clustering.html#k-means, 2025.

[7] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd ed., 2009.

[8] scikit-learn developers, "Spectral clustering — scikit-learn 1.7.2 documentation." https://scikit-learn.org/stable/modules/clustering.html#spectral-clustering, 2025.

[9] U. von Luxburg, *A Tutorial on Spectral Clustering*. Springer, 2007.

[10] F. R. K. Chung, *Spectral Graph Theory*. American Mathematical Society, 1997.

[11] scikit-learn developers, "Hierarchical clustering (ward method) — scikit-learn 1.7.2 documentation." https://scikit-learn.org/stable/modules/clustering.html#hierarchical-clustering, 2025.

[12] B. S. Everitt, S. Landau, M. Leese, and D. Stahl, *Cluster Analysis*. Wiley, 5th ed., 2011.

[13] scikit-learn developers, "Gaussian mixture models — scikit-learn 1.7.2 documentation." https://scikit-learn.org/stable/modules/mixture.html, 2025.

[14] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[15] scikit-learn developers, "Meanshift clustering — scikit-learn 1.7.2 documentation." https://scikit-learn.org/stable/modules/clustering.html#mean-shift, 2025.

[16] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1986.

[17] M. P. Wand and M. C. Jones, *Kernel Smoothing*. Chapman and Hall, 1995.

[18] scikit-learn developers, "Dbscan — scikit-learn 1.7.2 documentation." https://scikit-learn.org/stable/modules/clustering.html#dbscan, 2025.

[19] scikit-learn developers, "Silhouette coefficient — scikit-learn 1.7.2 documentation." https://scikit-learn.org/stable/modules/clustering.html#silhouette-coefficient, 2025. Accessed: 2025-12-01.

[20] scikit-learn developers, "Calinski-harabasz index — scikit-learn 1.7.2 documentation." https://scikit-learn.org/stable/modules/clustering.html#calinski-harabasz-index, 2025. Accessed: 2025-12-01.

[21] scikit-learn developers, "Davies-bouldin index — scikit-learn 1.7.2 documentation." https://scikit-learn.org/stable/modules/clustering.html#davies-bouldin-index, 2025.

[22] C. Shallahamer, *Forecasting Oracle Performance*. Apress, first ed., 2007.

[23] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system." https://doi.org/10.1145/2939672.2939785, 2016. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785–794.

[24] Oracle Corporation, "Statspack user's guide." https://www.oracle.com/technetwork/database/performance/statspack-129989.pdf, 2002.