Design and Implementation of Picasso 2.0 and A Glimpse of the Future Version

A Project Report
Submitted in Partial Fulfilment of the
Requirements for the Degree of
Master of Engineering
in
Computer Science and Engineering

By
Ravi Shetye

Computer Science and Automation
INDIAN INSTITUTE OF SCIENCE
BANGALORE – 560 012, INDIA

July 2009

Acknowledgements

I would like to thank my advisor Prof. Jayant Haritsa for allowing me to pursue this project under his guidance. I thank him, for his constant valuable guidance, support and encouragement during my tenure in DSL.

I would like to thank all the members of DSL for providing a stimulating and fun environment for work. I would like to thank rest of my friends at IISc who have made my stay at IISc memorable.

I thank my parents for their continued support throughout my career.

Abstract

Picasso is a tool being developed in the Database Systems Lab for visually analyzing the behavior of industrial-strength relational query optimizers. Database Systems Lab had decided to come up with a new version of the tool, called "Picasso 2.0". As an output of this project the lab released the tool "Picasso 2.0" in February 2009. This report talks about the design and implementation details of the new version.

The new major features implemented in Picasso 2.0 are "Approximate Diagram Generation" and "Dimension Specific Range and Granularity Diagram Generation". Also many minor features like "global coloring scheme", "consistency in the slice displayed in different tabs", "validity checks over the parameters the user selects", "new command line syntax" etc. have been added. Efforts have been put in to make Picasso 2.0 bug free and more robust than its predecessor.

This report talks mainly about the design and implementation of the new version of Picasso, "Picasso 2.0" and the bugs and loose ends fixed during the transition from the previous version to the new one.

"Picasso 1.0" and "Picasso 2.0" provide various features which enable the user to generate diagrams that throw light on the functioning of a query optimizer. However the user has to manually explore the diagram to find areas with unexpected, irregular fluctuation in estimated cost or cardinality. Also analysis of various plan diagrams across the query-templates or across different database engines is not supported by current versions of Picasso.

This report also talks about 'Data Mining' strategies proposed to be integrated in the future version of Picasso. These strategies would facilitate automated exploratory analysis to the user, thus enhancing the utility value of Picasso.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Picasso [10] is a tool developed in Database Systems Lab, which produces a variety of diagrams, showing the behavior of a database query optimizer.

These diagrams include the (compilation) Plan Diagram, which is a color-coded pictorial enumeration of the execution plan choices; the Compilation Cost Diagram, a visualization of the associated estimated plan execution costs; the Compilation Cardinality Diagram which is a visualization of the associated estimated query result cardinalities; and the Reduced Plan Diagram that shows the extent to which the original Plan Diagram may be simplified (by replacing some of the plans with their siblings in the Plan Diagram) without increasing the estimated cost of any individual query by more than a user-specified threshold value [6].

Picasso generates these diagrams that throw light on the functioning of the optimizer for a parameterized query template over the relational selectivity space. Given a query template, the grid resolution, the distribution at which the instances of this template should be spread across the selectivity space and the choice of query optimizer, the Picasso tool automatically generates the associated SQL queries, submits them to the optimizer to generate the plans, and finally produces Picasso diagrams [5].

Picasso is implemented completely in Java and should, in principle, operate in a platform-independent manner. It has been successfully tested on Windows (XP, Vista) and Linux/Unix. The client and server machines should support Java compilation and

execution while the client machine should additionally support 3D visualization. A few other third party visualization and database connection libraries like Java Database Connectivity (JDBC) drivers for the db engines, Java3D, VisAD, JGraph etc. are required for Picasso to function.

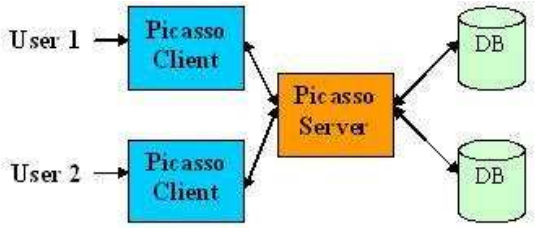


Figure 1.1: 3 Tier Architecture

A block diagram of the Picasso architecture is shown in Figure 1.1. Every request from the user is passed on from the Picasso client to the Picasso server, which handles communication with the database engine and the production of diagrams. The Picasso client is responsible for the visualization of these diagrams. The Picasso server communicates with the database engines through their JDBC interfaces, treating the optimizer's internals as black boxes. Picasso currently supports DB2, SQLServer, Oracle, Sybase and PostgreSQL.

A version of Picasso "Picasso 1.0" was released in May 2007 and has been made available on the lab's site since then as an open source software and a beta version was also available within DSL. We implemented the new functionalities using these projects as the base code.

We have improved the tool in a few aspects, mostly from the point of view of supporting the generation and visualization of "Approximate diagrams" and "Dimension specific Range and Resolution diagrams". Also the new tool has some supporting features like "Local and Global contribution indicators in the Legend Panel", "Consistency across the panels in the slice visualized ", "New command line syntax with more customization power to the user", "New sanity checks over the parameters the user provides" etc. Efforts have been put in making Picasso 2.0 bug free and more robust than the earlier versions.

Alongside the addition of new features, the bugs like memory leaks and the ToDos like improving efficiency of some part of code, in the beta version of Picasso were fixed.

A new version of Picasso, "Picasso 2.0" was released in February 2009 as an output of this project. Picasso 2.0 is available along with its predecessor on the DSL web site as an open source software.

After the release of the new version of Picasso, focus has been shifted to incorporate automated exploratory analysis within Picasso. Automated analysis would be such that it would facilitate the user to answer questions like :

- Within the Picasso diagram produced,
  - what are the areas which have unexpected behavior of some kind in their cardinality or cost estimations?
  - are there any operators which correspond to a certain behavior in the cardinality or cost estimation?
  - is there a particular tree layout which corresponds to a certain behavior in the cardinality or cost estimation?
  - is there a particular join ordering which correspond to a certain behavior in the cardinality or cost estimation?
- For a given database instance,
  - are there any tables or attributes which lead to high generation time for the plan diagram?
  - are there any tables or attributes which lead to greater number of plans, resembling finer choices made by the optimizer?
- For a given query template, across different engines, is there a common join ordering over a particular region?

Ideas have been proposed to incorporate 'automated exploratory analysis' within Picasso in future versions. We tried to answer the above mentioned questions using clustering strategies over the Picasso database at various levels and analyzing the clusters

produced by it. The experiments were conducted using the "Weka" data mining tool [7][8] developed at the University of Waikato.

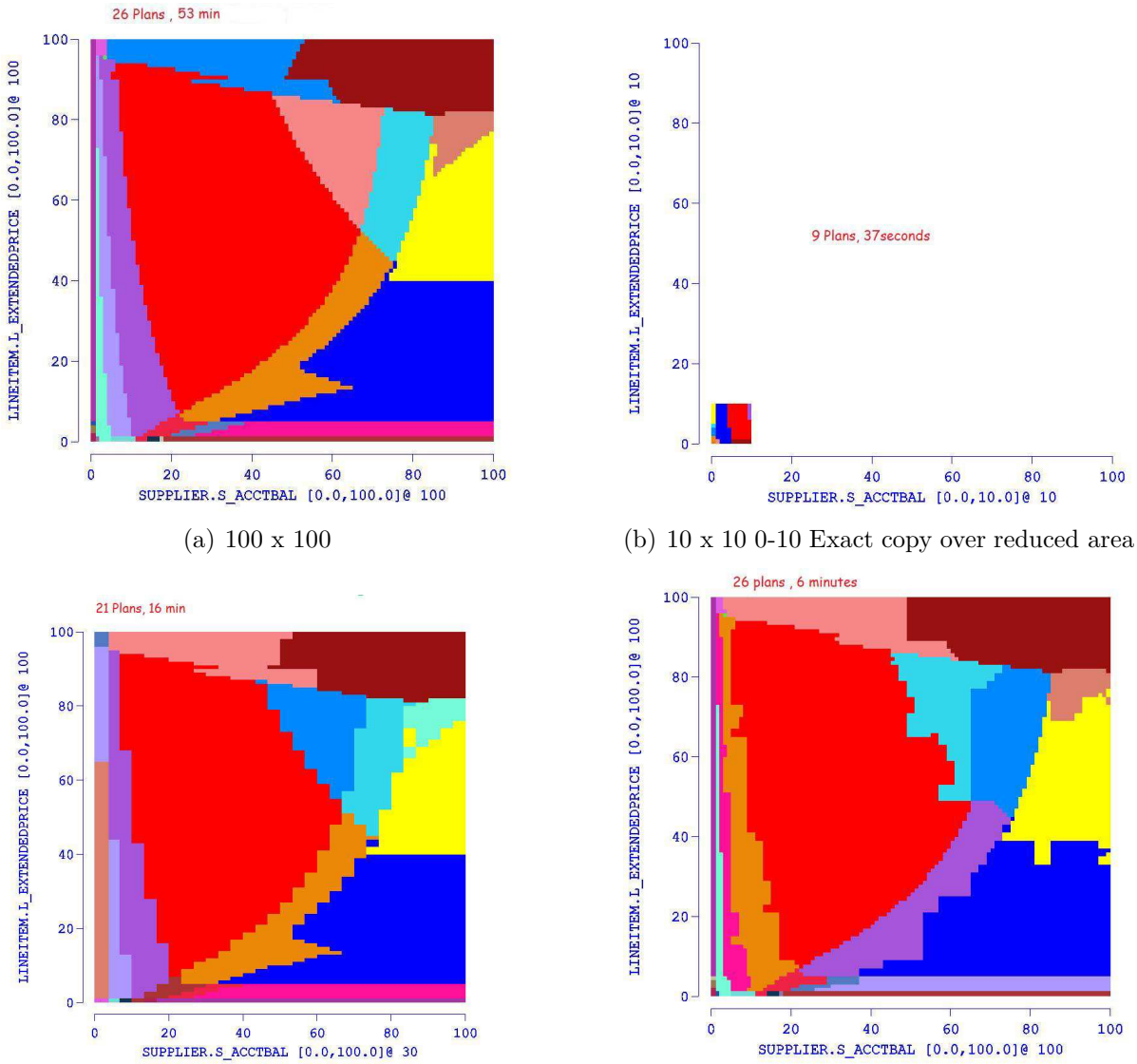
Section 1.1 talks about the motivation behind implementing the new features of Picasso 2.0. Sections 2.1 and 2.2 describe the Picasso data structures and the changes made to them in order to incorporate the new features. Section 2.3 describe the changes made to Picasso interface to facilitate creation of the modified data structures. Section 2.5 describes the changes made to the display of various Picasso diagrams. Chapter 3 describes the changes made in the Picasso command to be executed from the command line interface. Chapter 4 mentions the bugs fixed and other supplementary design changes made to Picasso client's design during the project. Chapter 5 mentions the programming effort which went in to create the new version of Picasso.

Further chapters talk about the ideas explored for integrating automated exploratory analysis with Picasso. Chapter 6 talks about the kind of analysis power we would like to provide to the user and shows the requirement for such kind of automated analysis by the user. Section 6.1 gives a short description of the current schema of the Picasso database and the changes in form of denormalization in form of logical views, required for efficient clustering. Chapter 7 describes the clustering algorithm used for providing the automated analysis. Chapter 8 concludes the report showing some results of the proposed automated analysis framework.

1.1 Motivation for Picasso 2.0

The main motivation behind adding the new features to Picasso is to address the high computation time requirement for the generation of high-dimension and/or fine resolution plan diagrams.

A typical plan diagram generated by Picasso 1.0 is shown in Figure 1.2(a). The "Dimension Specific Range and Granularity" functionality shown in Figure 1.2(b) and Figure 1.2(c) gives control to the user to generate diagrams only in the area of his interest and also with a higher resolution in the dimension of his interest. This saves time by not generating the diagram in the rest of the selectivity space and also generating the diagram



(a) 100 x 100

(b) 10 x 10 0-10 Exact copy over reduced area

(c) 30 x 100 Low Resolution on first dimension

(d) Approximate Diagram

Figure 1.2: Motivation for Picasso 2.0

at lower resolution on the secondary dimension.

"Approximate Diagram Generation" functionality shown in Figure 1.2(d) allows the user to generate an efficient approximation of the Picasso diagrams [4].

Chapter 2

Design of Picasso Client

Picasso client is used for getting the query template with the user desired settings for generation of the Picasso diagrams. Also, when the Picasso client receives the generated diagram from the Picasso server it displays them and allows the user to explore them interactively.

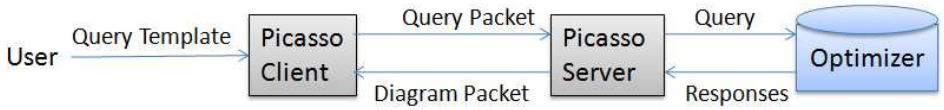


Figure 2.1: Query and Diagram Packet

The two data-structures used for communication between Picasso client and Picasso server are the 'QueryPacket' and the 'DiagramPacket'. Their previous structures and the changes made to them are discussed in the following sections.

2.1 QueryPacket

QueryPacket is the data structure, which the Picasso client generates based on the query template and the user specified setting in the "Settings Panel" Figure 2.3(a). The query-Packet is then sent to the Picasso server for further processing. The queryPacket of Picasso1.0 had the following information,

- queryTemplate:- The 'queryPacket' contains the entire 'Picasso query template', which is an SQL query that additionally features predicates of the form "relation.attribute :varies". These attributes are termed as 'Picasso Selectivity Predicates' (PSP). Each such query Template defines an n-dimensional relational selectivity space, where n is the number of PSP's.The response to the variation of selectivity of each of the PSP relations over the range 0 to 100% characterizes the optimizer behavior over this selectivity space.

```
select
        o_year,
        sum(case when nation = 'BRAZIL' then volume else 0 end) / sum(volume)

from

          (select YEAR(o_orderdate) as o_year,
                l_extendedprice * (1 - l_discount) as volume, n2.n_name as nation

          from part, supplier, lineitem, orders, customer,nation n1, nation n2, region

          where p_partkey = l_partkey and s_suppkey = l_suppkey and
                l_orderkey = o_orderkey and o_custkey = c_custkey and
                c_nationkey = n1.n_nationkey and n1.n_regionkey = r_regionkey and
                s_nationkey = n2.n_nationkey and r_name = 'AMERICA' and
                p_type = 'ECONOMY ANODIZED STEEL' and
                s_acctbal ≤ C1 and l_extendedprice ≤ C2

          ) as all_nations

group by o_year

order by o_year
```

Figure 2.2: Example Query Template: QT8

For example, consider QT8, the 2-D query template shown in Figure 2.1, based on query 8 of the TPC-H benchmark, with selectivity variations on the SUPPLIER and LINEITEM relations through the $s\_acctbal \leq C_1$ and $l\_extendedprice \leq C_2$ predicates, respectively. By varying the constants C_1 and C_2, queries are generated across the selectivity space.

- queryName:- The string by which the user can retrieve the query template and the associated Picasso Diagrams at a later point in time.

- resolution :- A single scalar quantity, because Picasso 1.0 would only produce dia-

grams with the same resolution on all axes.

- dimension:- The number of PSPs in the query template.
- execType:- Whether the diagram requested is a compile time or execution time diagram
- distribution:- Whether the query points are uniformly or exponentially distributed over the selectivity space.
- planDiffLevel:- Whether the plan trees should be compared at the 'operator' level or at the 'sub-operator or parameter' level.

In the newer version of Picasso the 'queryPacket' underwent the following changes

- The 'queryPacket' now has information of resolution for each dimension, instead of a single resolution for all dimensions.
- The 'queryPacket' has "StartPoint" and "EndPoint" for each dimension, which indicate the "Range" over the selectivity space for that dimension for which the user wants to generate the diagram.
- The 'queryPacket' also has a flag "ApproxDiagram" which is set if an approximate diagram is requested. Also it contains the error limits which are required by the approximation algorithm.

2.2 DiagramPacket

The diagramPacket underwent changes similar to the queryPacket. In addition to it one major change from Picasso 1.0 is that for higher dimensions (i.e. for dimensions greater than 3), earlier a single slice which was requested was sent to Picasso client by the Picasso server, but now the entire multi-dimensional packet is sent to the Picasso client and slicing of the diagram is done at the client side. This removes the delay of transferring each slice individually when it is requested for. In the new version, once a new slice of the

same diagram is requested, this request is fulfilled on the client side itself by choosing the correct entries corresponding to the slice from the entire multidimensional diagramPacket available at the Picasso client. To make the transfer of the entire multidimensional diagramPacket more efficient, we first compress the diagram and then create the diagram packet. The 'Picasso Server' uses a compression technique which is a combination of 'LZ77' and 'Huffman coding'. At the client side the diagramPacket is uncompressed, the multidimensional diagram is extracted and then the appropriate slice is displayed.

2.3 Creating QueryPacket

To create the queryPacket according to the new structure, various changes were made to the Picasso client's user interface. The user has been given the facility to select the type of diagram he wishes to generate, such as

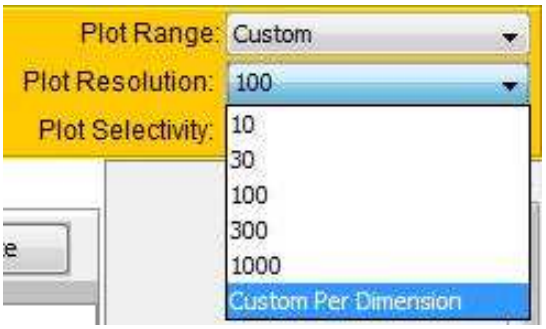
1. Traditional Picasso 1.0 Diagram
2. Dimension-specific Resolution Diagram
3. Dimension-specific Range Diagram
4. Approximate Diagram
5. Any combination of 2, 3 and 4

For this implementation the Picasso client's GUI underwent changes as described in Sections 2.3.1, 2.3.1 and 2.3.2.

2.3.1 Settings Panel

The 'Settings panel' as shown in Figure 2.3(a), is the panel where the user can select the database to connect to, the query template for which a diagram has to be generated, the distribution of the query points over the selectivity space, the optimization level, the resolution and the range for which the diagram has to be generated.

(a) Settings Panel



(b) Resolution Box

(c) Range Box

Figure 2.3: Settings Panel, Resolution Box, Range Box

In the older version of Picasso the "Plot Resolution" ComboBox as shown in Figure 2.3(b) had the values 10, 30, 100, 300, 1000 which indicated the resolution for all the dimensions of the diagram. We have inserted a new value "Custom Per Dimension" in the ComboBox, which the user needs to select if he wants to generate a 'Dimension Specific Resolution Diagram'. A point to note here is that even when we say 'Custom Resolution' the user is still constrained to use a value from the fixed set, but on per dimension basis. If the user chooses any of the numeric values from the ComboBox then that value is taken as the resolution over all the dimensions for further processing.

To select 'Dimension Specific Range Diagram', we have inserted another ComboBox "Plot Range" as shown in Figure 2.3(c), which has the values '0-100' or 'Custom'. '0-100' indicates that the Diagram will be generated over the entire selectivity space for all dimensions, the 'Custom' option indicates otherwise.

RangeRes Frame

If neither the 'Plot Resolution ComboBox', shown in Figure 2.3(b) nor the 'Plot Range ComboBox', shown in Figure 2.3(c) have 'Custom' as the value selected, then the queryPacket for the generation of the traditional Picasso diagram is created, and flow continues as per Picasso 1.0 . However if either of the ComboBoxes have 'Custom' selected then the RangeRes Frame as shown in Figure 2.4 is displayed.

In this frame the user can select the desired Range and Resolution for each dimension, and a queryPacket is created based on the values provided.

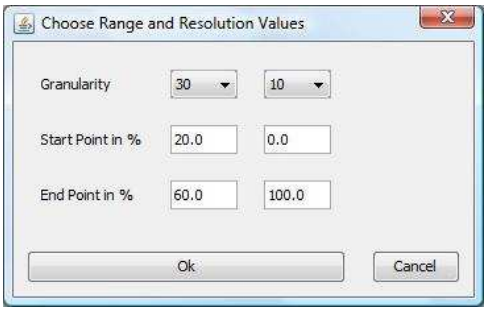


Figure 2.4: Frame for Selecting Dimension Specific Range and Resolution

2.3.2 Approx Frame

Once the queryPacket is generated it is passed to the server for estimating the time required for generating the exact diagram. The server sends this estimated time in form of a server packet back to the client.

On receiving the estimated time for exact diagram the client displays the time in the Approximation frame as shown in Figure 2.5 and gives the option for generating the approximate diagrams to the user.

If the user selects approximate diagram generation, then the frame expands allowing the user to select an approximation algorithm [4] and set the required parameter. The current version of Picasso has support for 2 approximation algorithms:

- RSNN Random Sampling Nearest Neighbor.

- GSPQO Grid Sampling Parametric Query Optimization.

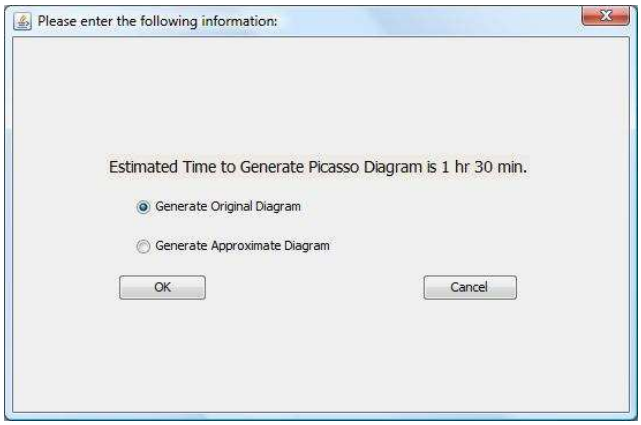


Figure 2.5: Frame Displaying the Estimated time and providing option for an approximate diagram

2.3.3 Parameter Validity Checks

Now that the new version gives a lot of customization power to the user, we have to implement various checks on the parameters the user sets, so that queryPacket does not contain any illegitimate values. A few of the checks which we make at the client side, so that 'queryPackets' with the wrong values get blocked at the client-side itself are:

- The values provided for range in the 'RangeRes Frame' Figure 2.4 must be numbers between 0 to 100.

- The endPoint value must exceed the startPoint value by at least '0.01'. This means that the highest granularity that the user can achieve is 1000 in the range of 0.01, which is equivalent to a virtual granularity of 100,000 over the entire selectivity range.

- One limitation which exists in current version of Picasso is that, the product of resolutions over all dimensions can not exceed the maximum size of 'int' as decided by Java. In Picasso 1.0 if the product of resolutions exceeds, the server shows an

error and hangs. This is a shortcoming of the version 1.0 and work is being done to remove this constraint. Till then instead of the server hanging, we do not allow the user to put values that may lead to a hang.

If any of these checks is not satisfied by the query packet, then an error message is displayed, no further process is done on this packet and the user is sent back to the 'QueryBuilder panel'.

2.4 Picasso Server

Once the final 'queryPacket' is sent to the Picasso server, it generates the required diagram over the selected range and at the selected resolution. Once the complete multi-dimensional diagram is generated it is packed into a 'diagramPacket'. This packet is then compressed as mentioned earlier and sent to the 'Picasso client'. This is in contrast to the earlier version of Picasso, 'Picasso 1.0' where the complete multi-dimensional diagram used to reside at the server-side only and the diagram packet used to carry information only about the requested slice.

2.5 Back to Picasso Client

Once the diagramPacket reaches the Picasso client, the client displays the diagram in the appropriate panel, with the help of Java3D and VisAD libraries. Changes were made to the various display panels to support the 'Dimension Specific Range and Resolution Diagrams'.

2.5.1 Plan Panel

Plan panel displays the plan diagram which has the color coded information of the plans selected by the optimizer for the particular values of the selectivity. In Picasso 2.0 we have placed the range and resolution information for a particular dimension, in the axis label for that dimension as shown in Figure 2.6.

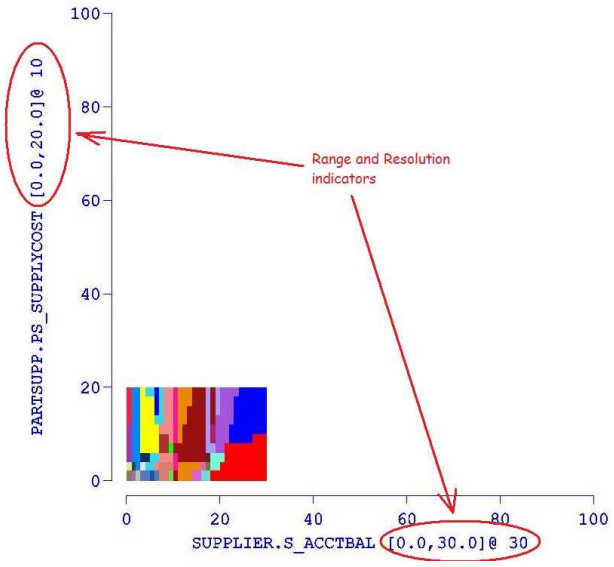


Figure 2.6: Change in the Display of the Diagram

To utilize the availability of the complete multidimensional diagram packet we provide the panel for slicing and pivoting the diagram as shown in Figure 2.7 at the bottom of the planPanel. By selecting appropriate values in this panel the user can pivot or slice the diagram. The pivoting and slicing functions had to be rewritten because these functions in Picasso 1.0 assumed 0-100% selectivity and same resolution on all the axis. For choosing the required slice for visualization the user has to click the "Set Dim Sel" Button as shown in Figure 2.7, upon which a frame as shown in Figure 2.8 for choosing the slice pops up, with only the valid selectivity values based on the range and resolution of the particular dimension.

Figure 2.7: Bottom Panel for Slicing and Pivoting

For efficient user interaction mouse clicks are active only for the area, for which the diagram has been generated, i.e. the white area in Figure 2.6 has been deactivated, hence any mouse clicks in that area will not trigger any processing. However we have still

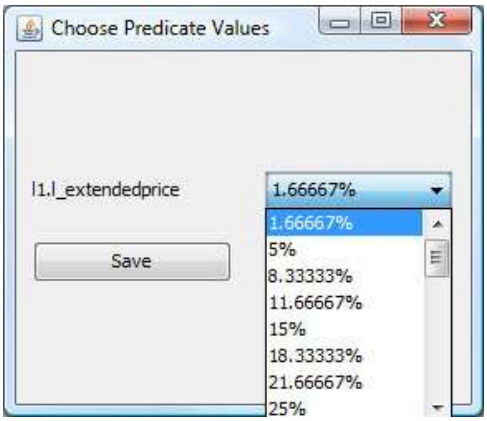


Figure 2.8: Frame for choosing a slice

maintained the previous functionality, i.e. the user can use any of the analysis power provided by Picasso 1.0 which are displayed in Figure 2.9, by clicking on the selectivity space for which the diagram has been generated. The functions which handled mouse interactivity also had to be rewritten for the previous version assumed 0-100% selectivity and same resolution on all the axis.

Now we will discuss about the implementation of the slicing and pivoting functionality and the 'mouse interactivity'.

Slicing and Pivoting

The main issue with the slicing and pivoting code of 'Picasso 1.0' was that it only considered same resolution on all axis and 0-100 range over the selectivity space for all dimensions.

The code for slicing had to be rewritten. The member points of the slice are now chosen giving consideration to the fact that the resolution might be different on each dimension.

The code of pivoting was modified such that now as the diagram is rotated, not only the data values but also the corresponding resolution and the ranges get rotated.

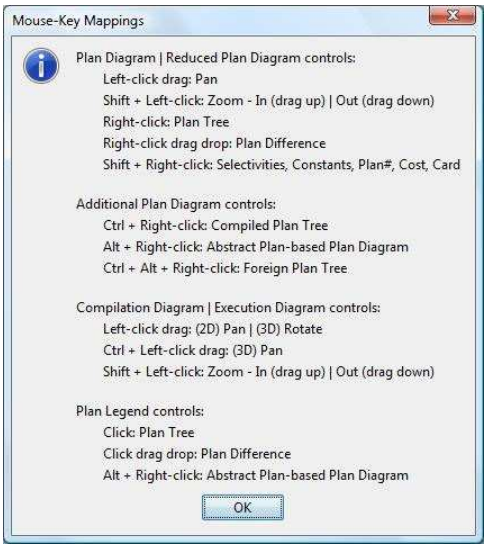


Figure 2.9: Mouse-Key controls for Picasso Diagrams

This ensures that after any number of rotations the data values when interpreted with the current resolution and range corresponding to the dimension will resemble the same multidimensional cube as before the rotation.

Mouse Interactivity

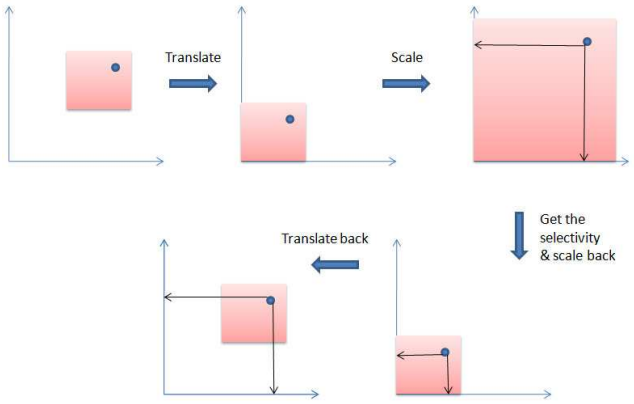


Figure 2.10: Implementation of mouse interactivity

The mouse interactivity of Picasso 1.0 also had the same drawback as the slicing functionality. It assumed the same resolution over all dimensions and the selectivity to be 0-100 over each dimension. To support mouse interactivity for dimension specific range and resolution diagrams we make modifications to the code which can be visualized as shown in the Figure 2.10. To correctly get the selectivity of the point clicked, and to exploit the reusability of the previous code we perform the following steps

1. We translate the point clicked, in each dimension by the StartPoint corresponding to that dimension. In 2-D analogy we bring the lower-left point of the selected space to the origin.

2. We scale the translated point, in each dimension by a factor of 100/(EndPoint corresponding to that dimension). In 2-D analogy we have now brought the upper right point of the selected space to the point resembling 100 percent selectivity over both the dimensions. We have now transformed the selected space into the standard 0-100 selectivity space.

3. We then calculate the corresponding selectivity with due consideration to the different resolutions over each dimension. A point worth noting is that the current resolutions are not the one the user provided, they also have been scaled. Hence in the entire calculation we consider the resolution to be the(user provided resolution)/(EndPoint-StartPoint corresponding to that dimension).

4. Once we calculate the selectivity, we now scale it down by the inverse of step 2.

5. We then translate it back to the original position.

At the end of this process we have the correct selectivity of the point where the mouse was clicked. Further processing is done based on which functionality from the figure 2.9 is demanded.

Changes on similar lines have been implemented for the other panels.

2.5.2 Reduce Plan Panel

The Beta version of Picasso had support for the following reduction algorithms [2][3]: Area Greedy, Cost Greedy, Cost Greedy with Foreign Plan Costing, SEER and Lite-SEER. We had to modify the implementation of these algorithms to support Dimension Specific Range and Resolution diagrams, however the basic structure of the algorithms remain unchanged.

The current version of Picasso supports Cost Greedy, CC-SEER and Lite-SEER based reductions.

2.5.3 Compilation Cost/Cardinality Panel

The Compilation Cost/Cardinality Diagrams have been modified on lines similar to the plan diagram. In addition modifications had to be made to normalize the cost/cardinality diagrams to the maximum cost amongst the query points lying in the selectivity space and fall within the range selected. A sample Compilation Cardinality Diagram is shown in Figure 2.11.

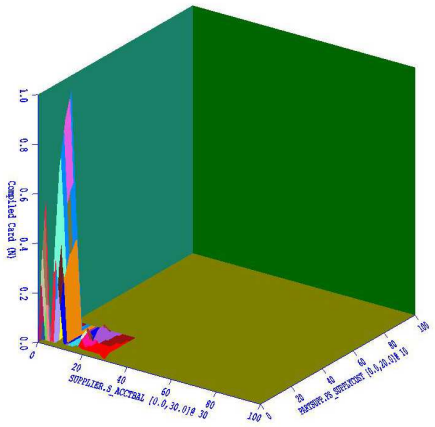


Figure 2.11: Compiled Cardinality Diagram

Legend Panel

To exploit further, the availability of the complete multi-dimensional diagram packet at the client side, the legend panel as shown in Figure 2.12 now shows the percentage participation by a particular plan globally in the entire multidimensional cube as well as locally in the selected slice.

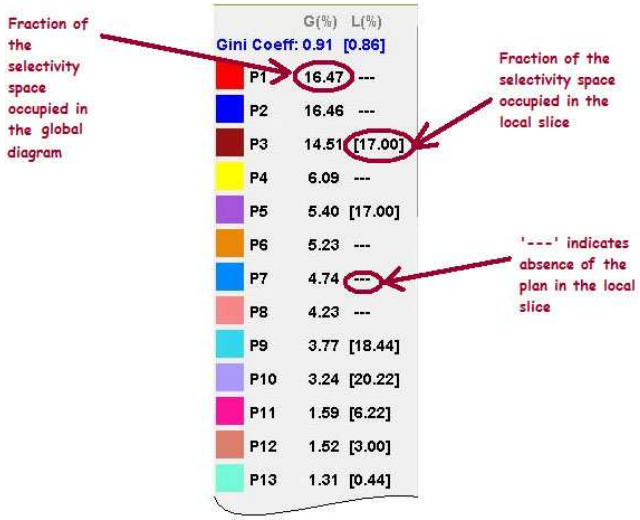


Figure 2.12: Legend Panel

Chapter 3

Command Line Interface

Apart from the Picasso Client interface described in the previous chapters, the Picasso Server can also be accessed directly through the command line. Changes were made to the syntax in order to support approximate diagram generation and dimension-specific range and resolution diagram generation. While making these changes care was taken that the previous syntax still works, to facilitate backward compatibility for the batch files containing the old syntax. If the user wishes to use the new features, the parameter list of the command must be started with '-R', the 'R' in the argument list indicates that the user is interested in generating a customized (R)ange or (R)esolution diagram.

The new syntax for running Picasso Client through command line is shown in Figure 3.

Basic format:

Without approximation:

PicassoCmd $<$ $ServerName$ $><$ $Port$ $><$ $DBConnection$ $><$ $OptLevel$ $><$ $QTID$ $><$ $QDist$ $><$ $DiagType$ $><$ $QTFile$ $><$ $Resolution$ $>$

Approximation:

PicassoCmd $<$ $ServerName$ $><$ $Port$ $><$ $DBConnection$ $><$ $OptLevel$ $><$ $QTID$ $><$ $QDist$ $><$ $DiagType$ $><$ $QTFile$ $><$ $Approx-algo$ $><$ $IdError$ $><$ $LocError$ $><$ FPC $><$ $Resolution$ $>$

Dimension specific Range and Resolution form

Without approximation:

PicassoCmd -R $<$ $ServerName$ $><$ $Port$ $><$ $DBConnection$ $><$ $OptLevel$ $><$ $QTID$ $><$ $QDist$ $><$ $DiagType$ $><$ $QTFile$ $>$ $\{<$ $Resolution$ $><$ $startPoint$ $><$ $endpoint$ $>\}$

Approximation:

PicassoCmd -R $<$ $ServerName$ $><$ $Port$ $><$ $DBConnection$ $><$ $OptLevel$ $><$ $QTID$ $><$ $QDist$ $><$ $DiagType$ $><$ $QTFile$ $><$ $Approx-algo$ $><$ $IdError$ $><$ $LocError$ $><$ FPC $>$ $\{<$ $Resolution$ $><$ $startPoint$ $><$ $endpoint$ $>\}$

$\{<$ $Resolution$ $><$ $startPoint$ $><$ $endpoint$ $>\}$ means that this triplet has to appear as many times as there are number of :varies in the Query template.

Figure 3.1: Command Line Syntax

Chapter 4

Bugs Fixed and Other Design Changes

During the process of developing Picasso 2.0, many bugs from the previous version were fixed. Few of these bugs are discussed below

- After a long series of operations (generating diagrams for more than 5 query templates without restarting the client), the Picasso Client module used to hang. On detailed analysis of the code of the previous versions, memory leaks were identified as the cause of the bug. The problem was solved by requesting a Java garbage collection.

- The dimension boxes in the bottompanel which are used for pivoting the diagram when selected, used to go behind the canvas used for drawing the diagrams. To correct this error the VisAD documentation was studied and the 'z-value' of the canvas was adjusted accordingly.

- The design of previous version of Picasso was such that, for a higher dimensional query template every time the user would toggle between the different types of Picasso Diagrams, the first slice used to be displayed. This has been changed to all the panels displaying the identical slice. So now, if the user chooses slice 'x' in the plan panel and switches to Compilation Cost Panel, then this panel will show a

Compilation Cost diagram corresponding to slice ‘x’.

Chapter 5

Programming Effort

The Picasso 1.0 code base is approximately 25K lines of code of which around 15K form the client and 8K form the server. The definitions of the common data structures and constants etc. form the remaining 2K lines of code. Most of the code was needed to be analyzed and about 15K lines of code were added additionally in Picasso 2.0 making it a total of approximately 40K lines of code. About 6K lines of code were added to the server of which 5K were for the approximation part and 1K were for the dimension specific range and resolution part. The latter part involved a lot of changes to the existing code whereas the former part was totally new. Approximately 10K lines of code were added to the client part the majority of which were to support:

- The dimension specific ranges and resolutions
- The slicing at the client side
- Global plan coloring etc.

Chapter 6

Automated Exploratory Analysis

The remainder of the report talks about the proposed framework for automated exploratory analysis to be integrated with the future version of Picasso.

Picasso 1.0 and Picasso 2.0 provide various functionalities to the user, by which user can generate variety of diagrams which reflect the behavior of the query optimizer in response to the query template and other parameters as provided by the user. However the user has to manually inspect these diagrams for any interesting or unexpected behavior of the query optimizer. After the release of the new version of Picasso, focus has been shifted to integrate automated exploratory analysis with Picasso. Such an automated analysis would work on the Picasso diagram produced and come up with interesting or unexpected area in the diagram.

As a part of the automated analysis we propose clustering strategies at various levels in the Picasso database. Clustering is a useful technique from the field of data mining, for grouping data points such that points within a single group/cluster have similar characteristics or are close to each other, while points in different groups are dissimilar. Hence such a clustering is helpful in identifying areas which have spikes or deviations from the expected monotonically increasing nature of the estimations over the selectivity space. Also clustering helps us to group together points of similar behavior which can be used to obtain a summarized description of the data set. This summarized description can then be used to answer many interesting questions about the Picasso diagram, the database

properties or the query optimizer.

Even though we allow clustering at different level of the Picasso database, still we maintain a common strategic framework. First we distinguish the attributes as measure attributes and the dimensional attributes. Measure attributes are the attributes which measure some value and can be aggregated upon. Dimensional attributes are the other attributes which define the dimensions on which the measure attributes are viewed. Further we cluster the data instances based on the measure attribute and later use the dimensional attribute to describe and analyze the clusters formed.

Before we discuss the exact clustering framework, let us briefly describe the design of the schema of the relations in the Picasso database and the relationships between them.We provide a short description about the Picasso database schema to get an idea about what information is available as a feature set for clustering.

6.1 Picasso Schema

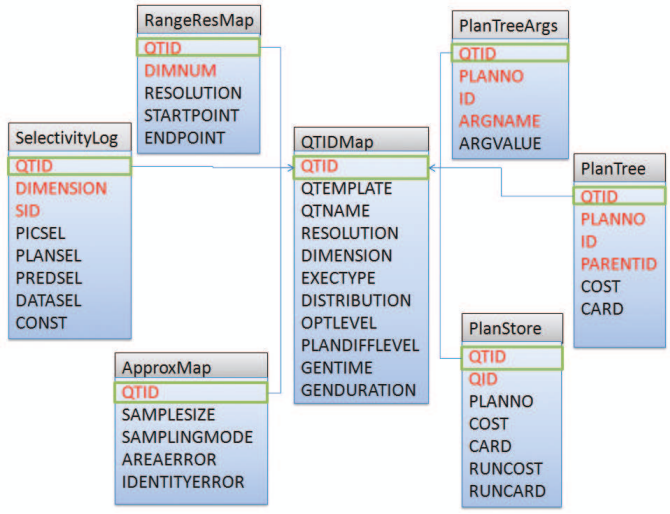


Figure 6.1: Picasso Schema

Figure 6.1 shows the relational schema of Picasso 2.0 and the next few paragraphs explain them in greater detail.

- PicassoQTIDMap stores information about each Picasso diagram generated such as the query template, query name, resolution, dimensions, optimization level, machine name, execution type, scale type etc.. A record in PicassoQTIDMap corresponds to a Picasso diagram. Each diagram is given a unique identifier called the query template identifies (QTID) which is the primary key for the relation.

- PicassoPlanTree stores the representative plan trees for each of the distinct plans in a Picasso diagram. Each record in this table represents a node in a plan. Every node has an ID as well as a parentID. This structure allows storing the plan tree as a relation. The QTID, plan number, the identifier of the node and the identifier of its parent together form the primary key of the relation.

- PicassoPlanTreeArgs stores the sub-operator level arguments for each node in the plan if any. Each tuple corresponds to one argument. A single node can have multiple sub-operator level arguments in which case there will be multiple tuples for a node. The set of all attributes forms the primary key for the relation because any proper subset of these set of attributes cannot uniquely identify a row.

- PicassoPlanStore stores the bulk of the Picasso diagram data including plan number, cost, cardinality for each of the points in the selectivity space. Each point in the selectivity space has a unique query ID (QID). The queries are fired in row major order and the queries are assigned in the same order. The QTID along with the query ID forms the primary key.

- PicassoSelectivityLog stores the Picasso selectivity, optimizer selectivity and actual selectivity values for each distinct point on a dimension. So there will be $\sum resolution_i$ number of tuples in this relation for a diagram. QTID along with the dimension and the point identifier forms the primary key of this relation.

- PicassoSelectivityMap maps each entry in PicassoPlanStore to entries in PicassoSelectivityLog.

- PicassoRangeResTab is a new relation created in Picasso 2.0 to support dimension

specific range and resolution. There is an entry for a Picasso diagram only if the resolution or the ranges on each dimension are different from default, the defaults being 0-100 for range and equal resolution on all the dimensions. There are dimension number of tuples in this relation for one diagram. The QTID and dimension together form the primary key for the table.

- PicassoApproxMap is also a new relation introduced in Picasso 2.0 to support approximate diagrams. This relation is essentially an extension to the PicassoQTIDMap. Like exact diagrams, the approximate diagrams also have an entry in the PicassoQTIDMap relation. But since we need to store additional information for approximate diagrams like the size of the sample required to generate the approximate diagram, the area and identity errors in the diagram and whether the foreign plan costing feature was turned on during the approximate diagram generation, we have a tuple in PicassoApproxMap for every approximate diagram generated. The attribute QTID forms the primary key of this relation.

Apart from the relations described above, we maintain separate relations to store abstract plans. For instance, SQL Server represents the Plan in XML. Abstract plans represented in XML are stored in the relation PicassoXMLPlan.

The QTID attributes in every relation (except PicassoQTIDMap) are foreign keys and refer the PicassoQTIDMap relation.

For further analysis we divide the attributes as those that are helpful for intra query template clustering and those that are helpful for inter query template clustering.

6.2 Intra Query-Template Clustering

As indicated in Figure6.2 we identify the following tables as the source tables for Intra Engine Intra Query-Template clustering :

- PicassoPlanStore
- PicassoSelectivityMap

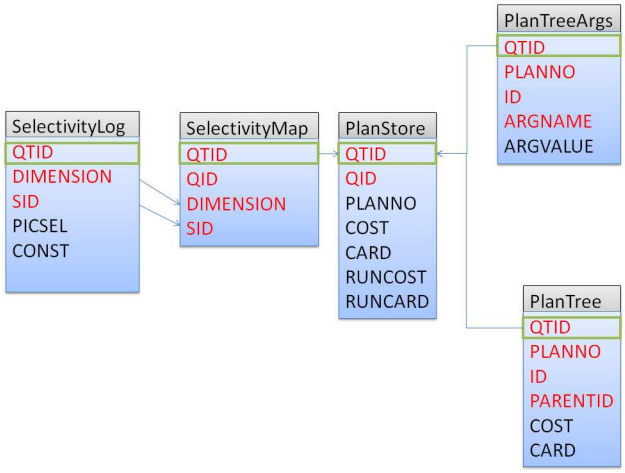


Figure 6.2: Intra Query-Template Clustering Schema

- PicassoSelectivityLog
- PicassoPlanTree
- PicassoPlanTreeArgs

Of these we use the PicassoPlanStore table to obtain the details about the measure attributes (compile time or run time cost or cardinalities). We use the PicassoSelectivityMap and PicassoSelectivityLog tables to obtain the selectivity information corresponding to each point in the Picasso diagram produced. We use the PicassoPlanTree table to obtain the plan tree structure corresponding to each point in the Picasso diagram produced. Ideally we should capture the entire layout of the plan tree. However such a layout is difficult to be provided as a feature to the clustering algorithm, so we just categorize the plan trees as bushy, left deep or right deep and provide it as a feature to the clustering algorithm. We use the PicassoPlanTreeArgs table to obtain the presence of a particular operator in the given plan tree.

Intra query template clustering helps us to answer questions of the following type :

- Within the Picasso Diagram produced, what are the areas which have unexpected behavior of some kind in their cardinality or cost estimations?

- Within the Picasso Diagram produced, are there any operators which correspond to a certain behavior in the cardinality or cost estimation?
- Within the Picasso Diagram produced, is there a particular tree layout which corresponds to a certain behavior in the cardinality or cost estimation?
- Within the Picasso Diagram produced, is there a particular join ordering which corresponds to a certain behavior in the cardinality or cost estimation?

6.3 Inter Query-Template Clustering

As indicated in Figure6.3 we identify the following tables as the source tables for Intra Engine Inter Query-Template clustering :

- PicassoQTIDMap
- PicassoPlanStore
- PicassoRangeResMap
- PicassoApproxMap

We obtain the measure attribute "generation time" from PicassoQTIDMap and "number of plans" and "maximum cost" from PicassoPlanStore. We use the PicassoQTIDMap table to obtain the other dimensional details of the Picasso diagrams produced. We use the PicassoRangeResMap and PicassoApproxMap tables to obtain the Range, Resolution and Approximation details of the Picasso diagrams produced.

Inter query template clustering helps us to answer questions of the following type :

- For a given database instance, are there any tables or attributes which lead to high generation time for the plan diagram?
- For a given database instance, are there any tables or attributes which lead to greater number of plans, resembling finer choices made by the optimizer?

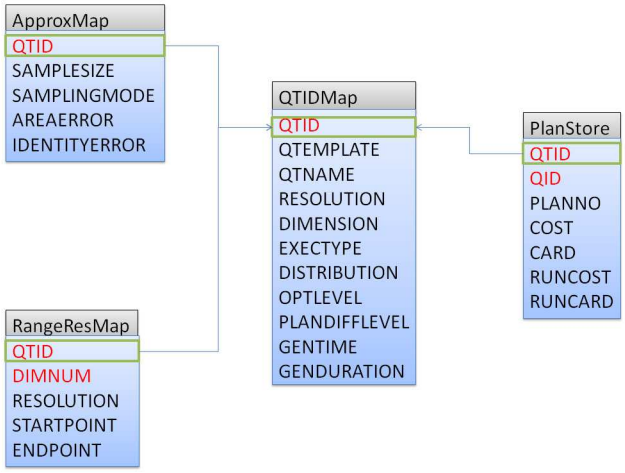


Figure 6.3: Inter Query-Template Clustering Schema

6.4 Inter-Engine Intra Query-Template Clustering

For Inter Engine Intra Query Template Clustering we use the same relational tables as used for Intra Engine Intra Query Template clustering. However we must note that the cost metric and the operators are not comparable across different database engines so we remove them from the measure and dimensional attribute lists.

Inter database engine clustering helps us to answer questions of the following type:

- For a given query template, across different engines, is there a common join ordering over a particular region?

The framework of automated exploratory analysis is described in Table 6.1.

Table 6.1: Clustering Framework

	Measure Attribute(s)	*Dimension Attributes*
Intra-Engine Intra Query Template	Cost, Cardinality	Selectivity, Operators, Tree layout, Join order
Intra-Engine Inter Query Template	# plans, Generation time, Maximum cost	Query template id, Resolution, Execution time, Distribution, Optimization level, Plan diff level
Inter-Engine Intra Query Template	Cardinality	Engine, Selectivity, Join order

For ease of clustering we need to de-normalize the above schema and store it. However de-normalization of schema brings into picture the tradition problems of redundancy and in some cases inconsistency. So we create logical views of the database which correspond to the de-normalized form of the set which contains the required attributes for clustering.

Chapter 7

Clustering Algorithms

In order to implement the above discussed clustering framework we propose the use of two clustering algorithms, each with their own pros and cons. The K-means clustering algorithm is described in Section 7.1 and the Expectation Maximization is described in Section 7.2

7.1 K-Means Clustering Algorithm

1. Begin
2. Initialize n, k, $\mu_1, \mu_2, ..., \mu_k$
3. Do
    a. Classify n samples according to nearest μ_i $1 \leq i \leq k$
    b. Recompute μ_i

   Until no change in μ_i
4. Return $\mu_1, \mu_2, ..., \mu_k$
5. End

Figure 7.1: K-Means Clustering Algorithm

The K-means clustering algorithm obtains K clusters which minimize the "Sum Squared Error" metric. The pseudo code is as shown in Figure 7.1.

Initially we choose k cluster centers $\{\mu_1, \mu_2, ..., \mu_k\}$ to coincide k randomly defined points inside the hypervolume containing the measure space. Then we assign each pattern to the closest cluster center and then recompute the cluster centers using the current cluster memberships. We repeat these steps for a fixed number of iterations or until the there is no further (significant) change in the cluster centroid.

K-means is a simple clustering method that, shows optimal results, when the data set have nearly globular natural clusters hidden in it [9]. K-means also has very low response time even for a 3-D Picasso diagram of resolution 100 on each dimension.

The problem with the clustering algorithm is that it needs the parameter 'k' to be provided by the user. Also when non globular cluster are naturally present in the data-set K-means fails to identify them efficiently.

7.2 Expectation Maximization Clustering Algorithm

Expectation Maximization (EM) is a statistical model that makes use of the finite Gaussian mixtures model. The algorithm is similar to the K-means procedure in that a set of parameters are re-computed until a desired convergence value is achieved.

A mixture is a set of N probability distributions where each distribution represents a cluster. An individual instance is assigned a probability that it would have a certain set of attribute values, given it was a member of a specific cluster.

A pseudo-code of the EM algorithm for bi-variate case is shown in Figure 7.2 and the generalization in it is mentioned in the subsequent paragraph.

For general case of multivariate normal density in d dimensions the function *f(x)* is written as,

$$f(x) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} e^{[-\frac{1}{2}(x-\mu)^t \Sigma^{-1}(x-\mu)]},$$

where x is a d-component column vector, μ is a d-component *mean vector*, Σ is the d-by-d *covariance matrix*, and $|\Sigma|$and Σ^{-1} are its determinant and inverse, respectively. $(x-\mu)^t$ denotes the transpose of $(x-\mu)$.

1. Guess initial values for the parameters : The mean μ, standard deviation σ and the cluster probability for each cluster.

2. Use the probability density function for a normal distribution to compute the cluster probability for each instance. In the case of a single independent variable with mean μ and standard deviation σ, the formula is:

   $$f(x) = \frac{1}{(\sqrt{2\pi}\sigma)} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

3. Use the probability scores to re-estimate the parameters.

4. Return to Step 2

Figure 7.2: EM Clustering Algorithm

The algorithm terminates when the log likelihood no longer shows significant increases. The likelihood computation is simply the multiplication of the sum of the probabilities for each of the instances. With two clusters A and B containing instances $x_1, x_2, \ldots, x_n$ where $P_A = P_B = 0.5$ the computation is:

$$[.5P(x_1|A) + .5P(x_1|B)] * [.5P(x_2|A) + .5P(x_2|B)] * \ldots * [.5P(x_n|A) + .5P(x_n|B)]$$

EM implements a statistical model that, shows optimal results, when the data set has dominant spikes in the distribution. EM does not require the parameter K to be passed to it, the algorithm determines the best number of clusters. The algorithm will converge to an optimal clustering; however, the optimization may not be global.

As the number of attributes increase the fitting of the mixture model becomes more complicated and increases the response time of the algorithm.

Chapter 8

Results and Observations

We experimented with the Picasso diagram database available in DSL. The clustering algorithms available in the WEKA [8] clustering utility were used to check the efficiency of the framework.

A sample template which had been clustered based on the framework as proposed in Section 6.2, is analyzed in the figures shown below. Figure 8.1 shows the scatter plot of the output of the EM clustering algorithm. The clustering information is color coded, i.e. instances belonging to the same cluster are assigned the same color. The estimated output cardinality of each point is plotted on the x-axis and the its corresponding estimated cost on the y-axis. The denser the cluster the more it follows the normal behavior, the sparse clusters represent the outliers or the irregularities. Based on this assumption we tag the clusters 2, 4, 6, 8 and 11 as interesting. As shown in Table 8.1 these clusters do resemble the irregularities in the cost and cardinality estimations as shown in Figure 8.2(a) and Figure 8.2(b). Table 8.1 shows a summary of each cluster with the dimensional attributes plugged in. It shows the clusters with the corresponding plans, the selectivity space it covers, and the operators which are present in the plans contained in the cluster. The sel_x and sel_y correspond to the selectivities of the bounding rectangle for the cluster. Detailed information can be obtained from a scatter plot similar to Figure 8.1 with sel_x and sel_y as the axis. For the columns corresponding to the operators '0' indicates complete absence of the operator in the entire cluster; '1' indicates total presence of the operator in the entire

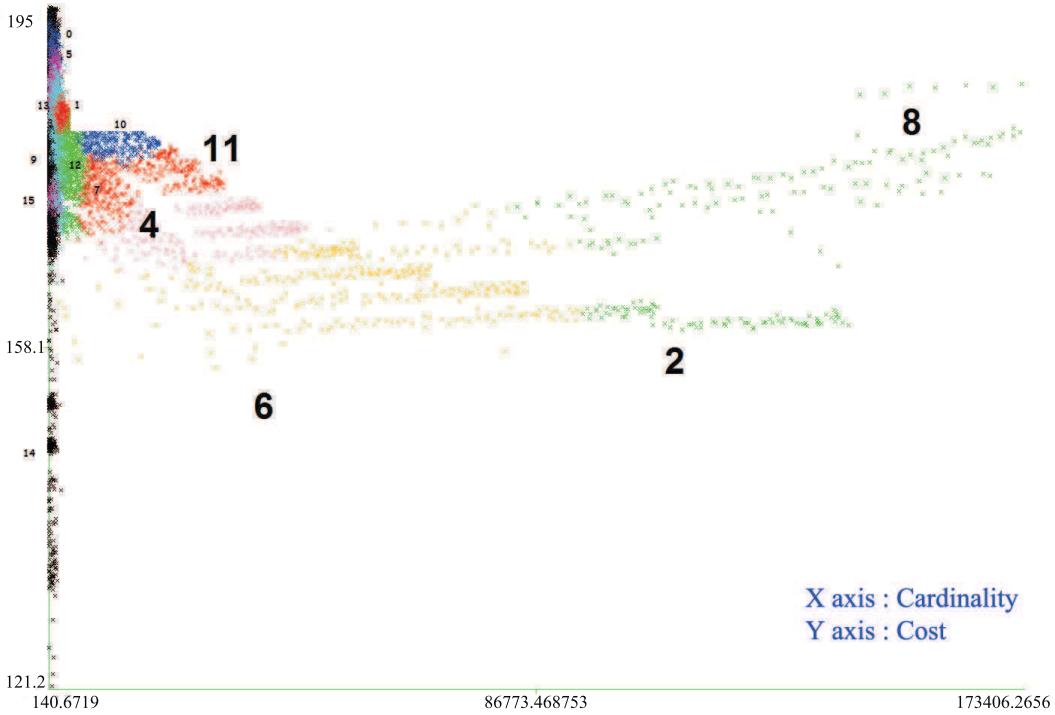
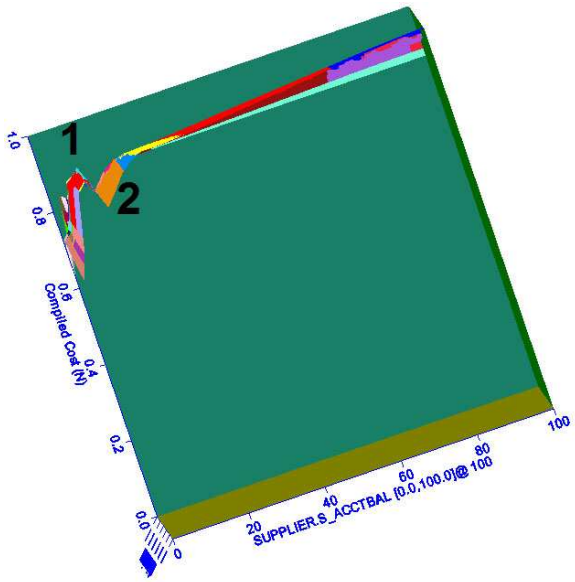


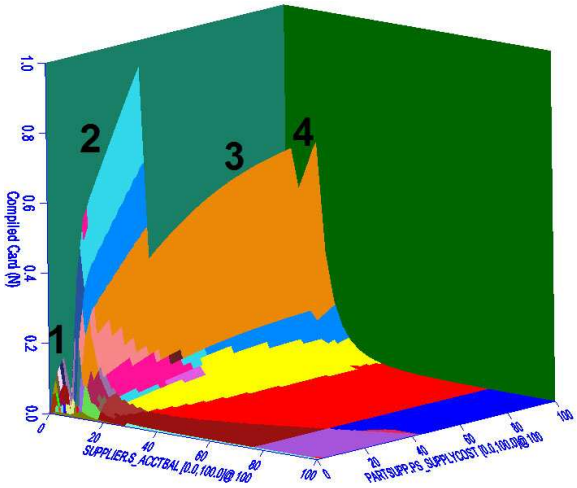
Figure 8.1: Cost vs. Cardinality Plot (TPCH : QT9)(EM Algorithm)

cluster; an entry of the form {x,y} indicates that in the 'x%' of the instances in the cluster the particular operator is absent and is present in the remaining 'y%' of instances. Due to space restriction, the operators which are common to all the cluster have been removed from the table. From the entries for sel_x and sel_y we can observe that the clusters which we tagged as interesting do correspond to the spikes in Figure 8.2(a) and Figure 8.2(b). Also it is interesting to note that even though clustering was carried out based on the dimensional attributes alone still we find dominance of the measure attributes over the clusters. So we may conclude that the presence or absence of a particular operator may be affecting the irregularities in the cardinality and cost estimation. An interested user of Picasso may explore this further.

Similar experiments were carried for higher dimensional templates, where in the clustering output was extremely useful in selecting the display dimensions and then selecting the slice selectivity for the other dimensions so that the slicing being displayed has irregularities in cost or cardinality estimations.



(a) Cost Diagram



(b) Cardinality Diagram

Figure 8.2: QT9 Cost and Cardinality Diagrams

Cluster	plans	sel_x range	sel_y range	bitmap	C.I.Seek	hash match agree	hash match partial agree	merge join	nested loop	parallelism	stream agree	table spool
0	30, 46	[57.5, 99.5]	[70.5, 88.5]	1	0	0	1	0	0	{93.36, 6.63}	1	0
1	35,	[67.5, 99.5]	[33.5, 47.5]	1	0	0	1	0	0	1	1	0
2	6, 15, 16, 25, 30, 31, 36	[1.5, 99.5]	[3.5, 11.5]	1	{28.02, 71.98}	{69.43, 30.57}	0	{99.36, 0.64}	{28.02, 71.98}	1	1	0
3	9, 10, 35	[0.5, 99.5]	[6.5, 47.5]	1	0	0	1	0	{99.62, 0.38}	1	1	0
4	5, 6, 15, 16, 17, 18, 36	[0.5, 99.5]	[9.5, 19.5]	1	{0.74, 99.25}	{68.17, 31.84}	{99.63, 0.38}	0	{0.74, 99.25}	1	{0.38, 99.63}	0
5	35, 46	[30.5, 99.5]	[17.5, 27.5]	1	0	0	1	0	0	1	1	0
6	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 15, 16, 17, 30, 31, 55	[1.5, 58.5]	[6.5, 9.5]	1	0	1	0	0	0	1	{1.9, 98.2}	0
7	5, 17, 18, 19, 30, 36, 39, 44	[1.5, 58.5]	[17.5, 27.5]	1	{80.75, 19.25}	1	{17.53, 82.47}	0	{80.75, 19.25}	1	{82.47, 17.52}	0
8	14, 30	[1.5, 96.5]	[6.5, 9.5]	1	0	1	0	0	0	1	{1.9, 98.2}	0
9	10, 11, 20, 35	[0.5, 42.5]	[6.5, 78.5]	{0.4, 99.6}	0	0	1	1	1	{0.4, 99.6}	1	0
10	31, 39	[52.5, 99.5]	[19.5, 29.5]	1	{86.25, 13.75}	{13.75, 86.25}	{13.75, 86.25}	0	{86.25, 13.75}	1	{86.25, 13.75}	0
11	16, 30, 31, 39, 45	[31.5, 99.5]	[17.5, 20.5]	1	{20.38, 79.62}	{77.25, 22.75}	{86.26, 13.74}	0	{20.38, 79.62}	1	{13.74, 86.26}	0
12	10, 19, 35, 39	[1.5, 99.5]	[6.5, 39.5]	1	0	{49.23, 50.77}	1	0	0	1	{50.77, 49.23}	0
13	35	[21.5, 99.5]	[44.5, 69.5]	1	0	0	1	0	0	1	1	0
14	0, 10, 11, 12, 15, 20, 21, 22, 23, 24, 26, 27, 28, 29, 32, 33, 34, 35, 37, 38, 40, 41, 42, 43, 46, 47, 48, 49	[0.5, 99.5]	[0.5, 99.5]	{2.96, 97.04}	{87.41, 12.59}	{97.04, 29.64}	{13.36, 86.64}	{99.42, 0.58}	{87.41, 12.59}	{39.32, 60.69}	1	{96.84, 3.16}
15	10, 35	[1.5, 71.5]	[6.5, 54.5]	1	0	0	1	0	0	1	1	0

Table 8.1: QT9 : Clusters with Dimensional Attributes

Chapter 9

Conclusions and Future Work

We conclude by saying that Picasso 2.0 has many more useful features than those that Picasso 1.0 had and we believe that it has a utility value greater than Picasso 1.0. As shown in this report, there is scope for providing automated analysis power to the Picasso user. In future we would like to convert the proposed idea into a finished product and make it available as a third party tool along side Picasso. We also would like to provide Picasso user with incremental diagram generation power, so that the user need not wait for entire diagram to be generated before it is displayed.

References

[1] N. Reddy and J. Haritsa, "Analyzing Plan Diagrams of Database Query Optimizers", Proc. of 31st Intl. Conf. on Very Large Data Bases (VLDB), August 2005.

[2] Harish D., P. Darera and J. Haritsa, "On the Production of Anorexic Plan Diagrams", Proc. of 33rd Intl. Conf. on Very Large Data Bases (VLDB), September 2007.

[3] Harish D., P. Darera and J. Haritsa, "Identifying Robust Plans through Plan Diagram Reduction", Proc. of 34th Intl. Conf. on Very Large Data Bases (VLDB), August 2008.

[4] A. Dey, S. Bhaumik, Harish D. and J. Haritsa, "Efficiently Approximating Query Optimizer Plan Diagrams", Proc. of 34th Intl. Conf. on Very Large Data Bases (VLDB), August 2008.

[5] Mohammad Aslam, *ME Thesis, CSA*, May 2006.

[6] Abhijit Pai, *ME Thesis, CSA*, May 2008.

[7] Ian H. Witten and Eibe Frank (2005) "Data Mining: Practical machine learning tools and techniques", 2nd Edition, Morgan Kaufmann, San Francisco, 2005.

[8] Weka 3: Data Mining Software in Java, *http://www.cs.waikato.ac.nz/ ml/weka/*

[9] Jain, A. K., Murty, M. N., and Flynn, P. J., "Data clustering: A review", ACM Computing Surveys **31**, (1999).

[10] Picasso Database Query Optimizer Visualizer, *http://dsl.serc.iisc.ernet.in/projects/PICASSO/picass*

[11] DSL Publications, *http://dsl.serc.iisc.ernet.in/publications/publications.html*