Report for CS8359-03 Visual Analytics and Machine Learning

Wei Fan
Instructor: Matthew Berger

Abstract

FECBench is an open source framework, which guides providers in building performance interference prediction models for their services without incurring undue costs and efforts. It builds models for target applications based on 106 applications' performance data by putting different system pressure on them and do prediction on the target application's performance. Due to the diversity of dimensions, understanding the dataset is highly significant to researchers in designing and verifying the model and target applications. Dimension reduction has been a sufficient way to give insight to high dimensional data. Among a variety of dimension reduction methods, Principle components analysis (PCA) has been widely used as a mathematical technique to show dimension reduction result. Thus, in order to provide users a clear understanding and an accurate analysis of the FECBench dataset, I have developed a system that visualizes the results of principal component analysis using multiple coordinated views and a set of user interactions.

Introduction

Context

FECBench (Fog/Edge/Cloud Benchmarking), which is an open source framework comprising a set of 106 applications covering a wide range of application classes that guides providers in building performance interference prediction models for their services without incurring undue costs and efforts via the following contributions [1].

The goal of FECBench is to minimize the efforts for developers in building interference-aware performance models for their applications by providing them a reusable and extensible knowledge base. To that end, FECBench comprises an offline stage with a set of steps to create a knowledge base followed by an online stage [1]. Developers can use the same offline stage process to further refine this knowledge base.

Goal

It's important to get a deep insight in the relation between system pressure and application performance no matter in the stage of building model or in evaluating performance for a target application. To provide a clear visualization on a deep understanding and an accurate analysis of the FECBench dataset, developing a system that visualizes the results of principal component analysis using multiple coordinated views and a set of user interactions can be a sufficient design.

Data

The data is from a time series dataset about system pressure and performance of a running target application. There are about 300 data items per application. Each data item contains a timestamp, some important system metrics to represent the system pressure which the application is suffering at that time, and the corresponding performance. We use an attribute called 'latency90' to evaluate the application performance, which means 90 percentile of the latency. The latency means the time it takes for a data packet be transmitted from server to client.

Specific attributes and description are descripted in the following table.

Attribute	DataType	Description
Date	Timestamp	HH:MM:SS
Name	String	Application's name
System bandwidth	float	Resource pressure – a system metric
Memory bandwidth	float	Resource pressure – a system metric
Context switch	float	Resource pressure – a system metric
Cpu percent	float	Resource pressure – a system metric
Disk io	float	Resource pressure – a system metric
Host memory	float	Resource pressure – a system metric
Host network	float	Resource pressure – a system metric
Latency90	float	Performance - a system metric
Latency	float	Performance - a system metric

Task

According to the overall goal and what data we get, task can be specifically divided into the following subtasks.

a. Clear view on system pressure

Analysis on system pressure plays a significant role in building models. So we need to provide straightforward views for raw pressure metrics data, eigenvectors of these system metrics and PCA results on them. Also, it's better to know if the two metrics are positive correlated and how they correlate with each other.

b. View on performance

'Latency90' is the metric which we use to evaluate application performance in this case. The higher the latency is, the lower the performance is. So a clear visualization on 'latency90' curve should be included to show the variance of performance at different time steps.

c. Relation between performance & system pressure

FECBench has been looking at how resource pressure influences application's performance. So understanding relation between performance and pressure has great value for reference when building the model. We need to give plots to answer questions like 'what causes this time step in high latency? ' or 'what's the bottleneck of this application? '.

d. Time period comparison in system pressure & application performance

Since the dataset contains time series data, there should be some time periods with high latency and some with low. Doing comparison on pressure and performance between these different time intervals not only allows user understand the causes on different latency values, but make correlation of different metrics a bit clearer.

Model

There are a bunch of dimension reduction methods and I choose Principle components analysis, which is defined as a linear transformation that transforms the data to a new coordinate system such that the greatest variance by some projection of the data comes to lie on the first principal component , the second, and so on. The reason I choose PCA to do dimension reduction is that each system metric is quantitative data and we need to show user which is the application bottleneck, and PCA is easy to implement with sklearn and returns result as we expect.

a. PCA

PCA is widely used for reducing dimensions for high dimensional data. It is defined as a linear transformation that transforms the data to a new coordinate system [2].

It first calculates the covariance matrix *X* of data points, eigen vectors and corresponding eigen values. Then it sorts the eigen vectors according to their eigen values in decreasing order. After sorting, it then chooses first k eigen vectors as the new k dimensions. The last step is to transform the original n dimensional data points into the k dimensions [3].

Here is equation for getting the first component.

$$w_{(1)} = \underset{||w||=1}{\operatorname{argmax}} \left\{ \sum_i (t_1)_{(i)}^2 \right\} = \underset{||w||=1}{\operatorname{argmax}} \{ \sum_i (x_i \cdot w)^2 \}$$

For further components,

$$\widehat{X_k} = X - \sum_{s=1}^{k-1} X w_{(s)} w_{(s)}^T$$

$$w_{(k)} = arg \max_{||w||=1} \{ ||\widehat{X_k} w||^2 \} = argmax \{ \frac{w^T \hat{X}_k^T \widehat{X_k} w}{w^T w} \}$$

The most popular way of showing PCA result is transforming the original data into 2 dimensions and plot a scatter plot. However, the components ratio of principle components and eigenvectors are also significant to depict the relation between system metrics in this case. So I took both the components and transformation results into account.

b. Pearson correlation coefficient

The Pearson correlation coefficient is a measure of the strength of the linear relationship between two variables. It has a value between +1 and −1, where 1 is total positive linear correlation, 0 is no linear correlation, and −1 is total negative linear correlation. In my case the variables are system metrics and we can see the correlation between them by calculating Pearson correlation coefficient [4].

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}$$

$$\mathrm{Cov(X, Y)} = \mathrm{E[XY]} - \mathrm{E[X]E[y]}$$

Related Work

There are many papers regarding to either applying PCA transforming result to visualizations or grasping how PCA works. Among them, I choose paper 'iPCA: An Interactive System for PCA-based Visual Analytics' as a project design baseline.

This paper, aims to assist the user in better understanding and utilizing PCA for analysis rather than just take the transforming result with treating PCA as blackbox. It shows design prototypes for using parallel coordinates to view raw high dimensional data and corresponding eigenvectors, a scatter plot to project the dataset on the first two principle components [2]. It also provides bunch of interactive tools for a better understanding of the PCA model.

This paper gives valuable references for me on the plot design to some extent. I used some ideas of the view design from it and adjusted its interactions since the goal of iPCA is making the algorithm transparent but mine is not.

Design Detail

Overall description

The project is a web project, based on Flask and JQuery to pass values from backend to frontend. There's no database used for this demo and the framework is on Model-Template-View style.

The main libraries I used are D3.js, Plotly.js and Tabulator. To break the limitations these libraries bring, I made composite use of them and modified them to reach my expectation.

The whole visualization consists of 6 views: Performance view (A), Correlation view (B), Data view (C), Eigenvector view (D), Projection view (E), and Principle components view (F).

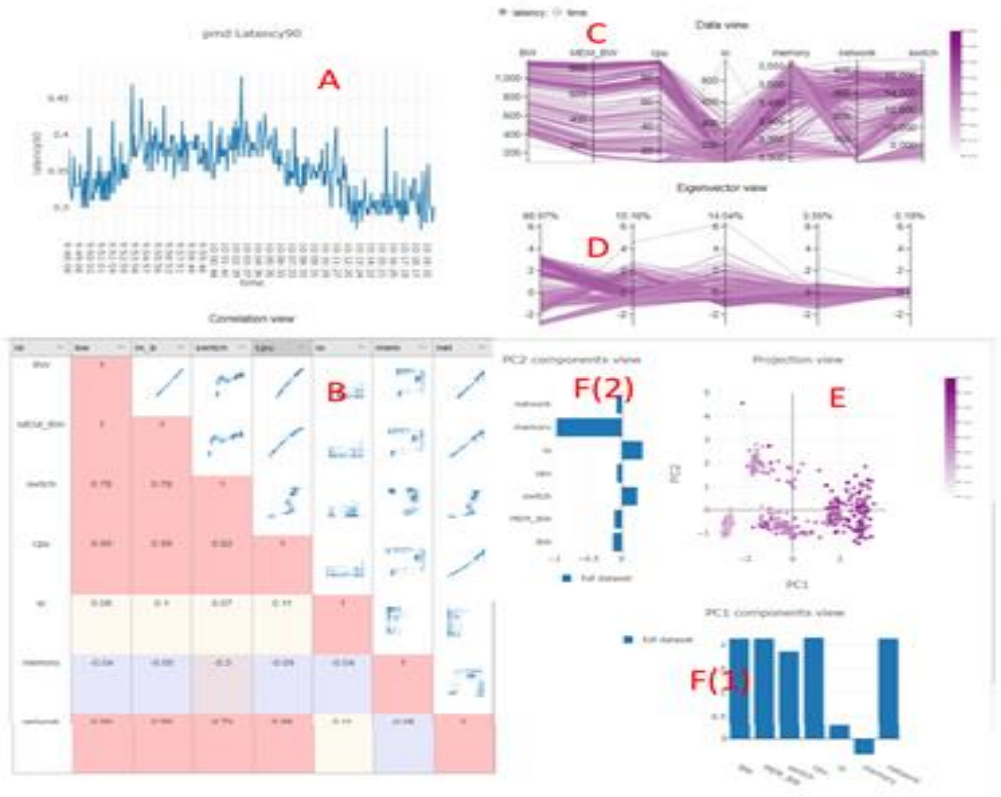


Figure 1 – Overall view

The specific encoding and interactions of each view will be introduced in the following sections.

Performance view (A)

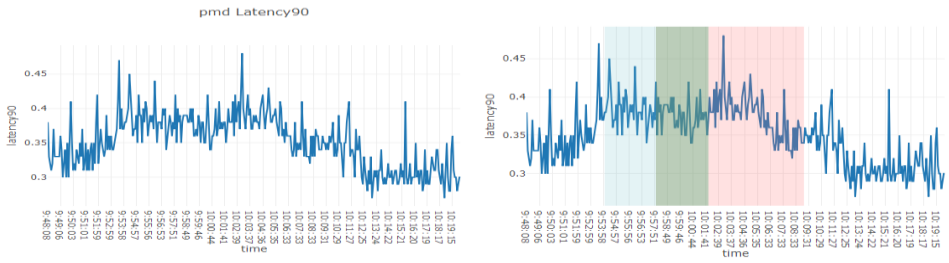


Figure 2 – Performance view

- Lib
  - Plotly.js + d3.js
- Data
  - Full dataset [‘latency’, ‘time’]
- Encodings
  - X: time series (timestamp)
  - Y: latency (performance)
  - Background color: range selected
- Interactions
  - Range selection/cancel. Overlap selections are allowed. Linked with View (B, C, D, E, F).

The application’s performance is evaluated by a single metric, latency. And the dataset is a time series dataset, so using a line chart is a good way to give insight to performance. The data encoded here is the time and latency attributes from the raw dataset. X axis is time, Y axis is latency. User can drag ranges for comparison because you can see there are some time periods with high latency, and some are low. Thus it’s important to allow user to compare with these intervals to see what happens. And the ranges user select are marked with different background colors. Selecting time periods influences all plots. View (B, C, D, E, F) will all update according to the time intervals selected in View A.

Correlation view (B)

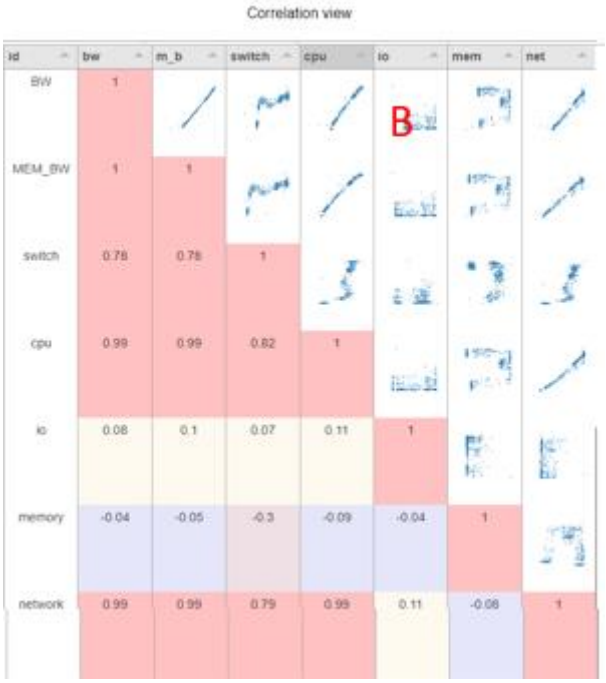


Figure 3 – Correlation view

- Lib
  - Tabulator + plotly.js + d3.js
- Data
  - Person correlation coefficient value on the selected ranges
- Encodings
  - Table
    - Row/column: system metrics
    - Cell value: Person correlation coefficient value + inline plots
    - Cell color: coefficient value.
    - Color scale: cold/warm colors. (cold color: <0; warm: >0)
  - Inline plots
    - X: row metric
    - Y: column metric
- Interactions
  - Be linked with performance view
  - In-graph: zoom

The correlation view (B) is a table with inline scatter plots to give a clear insight in correlation between system resources. The data encoded is Pearson correlation coefficient value on the selected ranges. Default is based on the full dataset. As you can see, the rows and columns are metrics, cell values are numbers and scatter plots. For the cells that contains numbers, the values are Pearson correlation coefficient value. I use cold colors for background to represent negative values and warm colors to show positive ones. The colder or warmer the cell is, the smaller or larger the value is. As for the inline scatter plot, x-axis is metric value in row and y-axis is metric value in column.

The Pearson correlation is important in PCA and people need to know the correlation between two metrics. Inline scatter plots give a straightforward view about the correlation. This idea is from iPCA paper.

Data view (C)

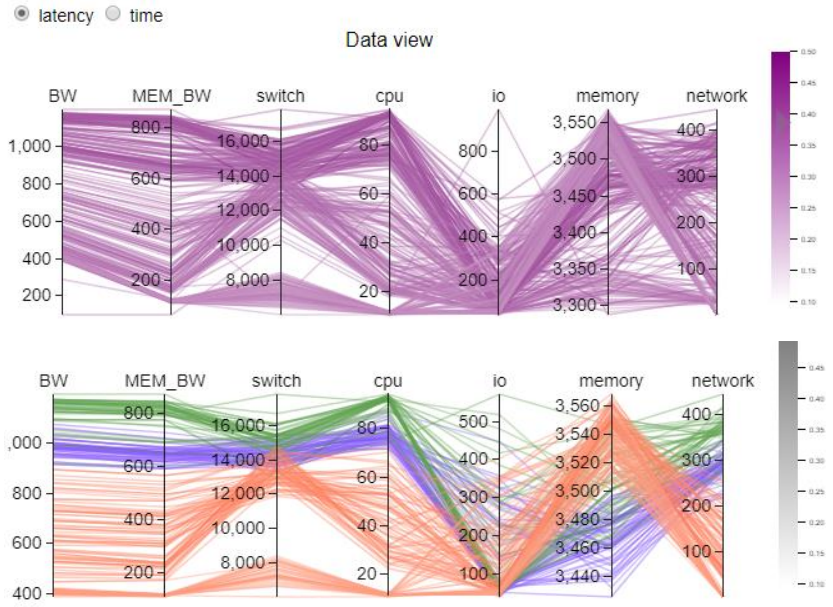


Figure 4 – Data view

- Lib
  - D3.js - parallel-coordinates(syntagmatic) && Mostapha Sadeghipour Roudsari
- Data
  - System metrics attributes from raw dataset on selected time periods
- Encodings
  - Coordinate: system metric
  - Brightness: latency/ time (user can choose) – gradient color scale
  - Color: time intervals selected by user

    Time period selected – discrete color to represent different time period
- Interactions
  - Brush on dimension
  - Click on single data item – find outlier
  - Hover
  - Linked with View (A, D, E)
  - Select which attribute to be represented by color

The data view is designed as a parallel coordinate plot. Data used here is raw data. If user select time periods on the performance plot, this data view will show selected data and use different colors to group them. The illumination color scale, like from light purple to dark purple, light red to dark red, represents the latency.

The data view allows user to select either multiple data items or single data item by brushing on coordinate or clicking on lines. If lines are selected, same data items in View D, E, A will be highlighted. (Screenshot can be found in Results Section). User can cancel selection by double

clicking. I also set a radio for user to switch the color encoding from the default ‘latency’ to attribute ‘time’, with the color bar updates and be well scaled.
Parallel coordinate plot is good for viewing high dimension raw data. I choose latency and time in illumination color scale because user can observe something like ‘high latency is always along with a high cpu precent’ or ‘with time going, the network metric decreases’, which is really helpful in analyzing relation between application’s performance and resource pressure.

Eigenvector view (D)

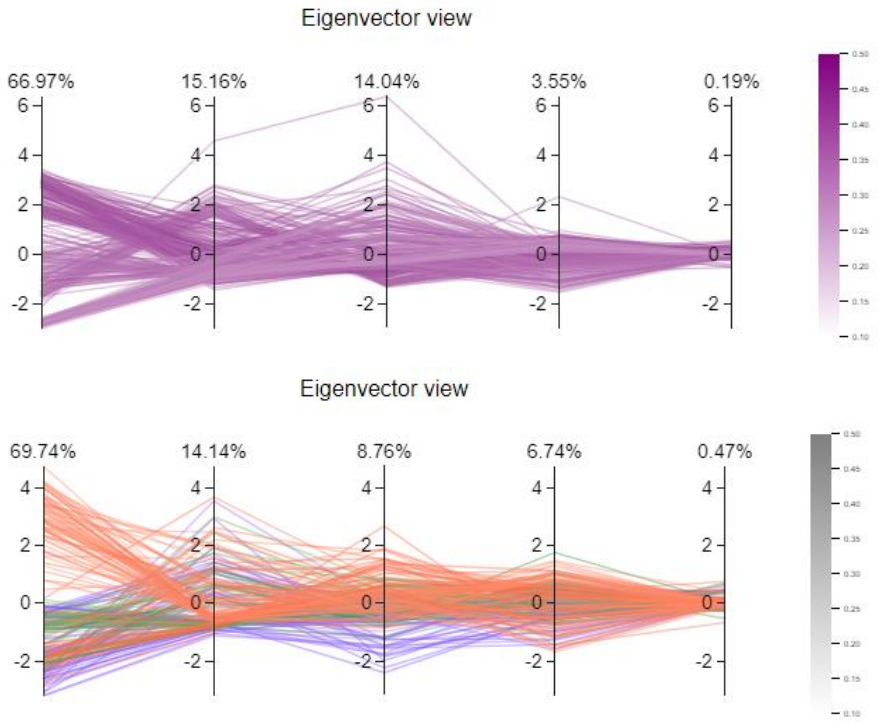


Figure 5 – Eigenvector view

- Lib
  - D3.js - parallel-coordinates(syntagmatic) & Mostapha Sadeghipour Roudsari
- Data
  - PCA.explained_variance_ratio
- Encodings
  - Coordinate: ratio
  - Line: eigenvector for each data item
  - Illumination: latency/ time (user can choose) – gradient colorscale
  - Color: time intervals selected by user
    Time period selected – discrete color to represent different time period
- Interactions
  - Brush on dimension
  - Click on single data item – find outlier
  - Hover
  - Linked with View (A, C, E)
  - Select which attribute to be represented by color

Eigenvector view uses PCA explained_variance_ratio. The coordinate is ratio, which is in order. Each coordinate is in common scale. Each eigenvector is treated as a dimension in this parallel coordinates view, and every data item is drawn as a line. The calculated eigenvectors and their eigenvalues are displayed in a projected parallel coordinates visualization, with eigenvectors ranked from left to right by dominance.

It has same color encoding and same interactions with data view, which allows user to select either multiple data items or single data item by brushing on coordinate or clicking on lines. If lines are selected, same data items in View C, E, A will be highlighted. (Screenshot can be found in Results Section). User can cancel selection by double clicking. The radio above View C also works for this eigenvector view to switch the color encoding from the default 'latency' to attribute 'time', with the color bar updates and be well scaled.

This plot tries to explain how to get the principle components in detail. So it's important for PCA visualization.

Projection view (E)

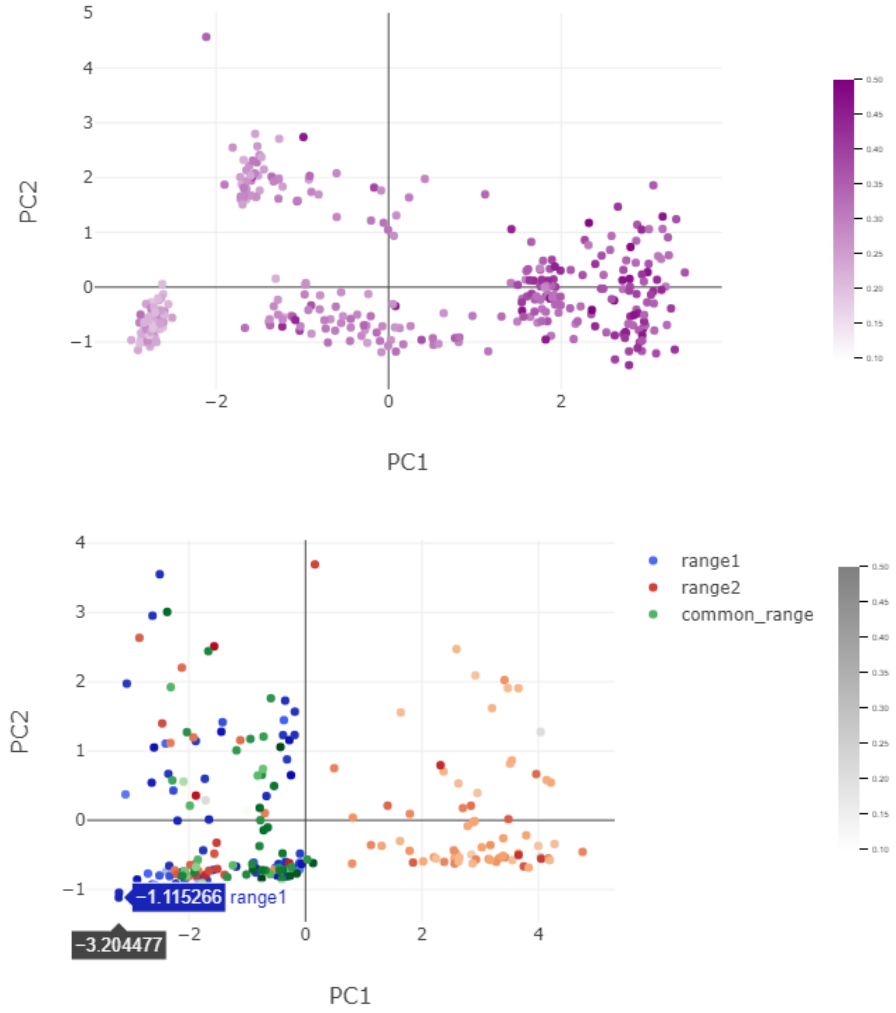


Figure 6 – Projection view

- Lib
  - Plotly.js
- Data

    - PCA first 2 principle components projection
- Encodings
    - X: PC1
    - Y: PC2
    - Color: time period selected
    - Illumination: latency/time (can switch)
    - Opacity: selected or not (by parallel plots)
- Interactions
    - Zoom

The projection view uses a scatter plot to show the 2 dimension PCA project result. I draw it because it's the most direct result we get from PCA. User can get the most 2 important composite components which describe the high dimensional dataset best. Scatter plot provides a clear view on the projection.

X-axis refers PC1 value and y-axis refers PC2 value. Color represents time period selected by user in the line chart. The illumination in each group represent the attribute 'latency' or 'time'. If user brush or select data items in parallel coordinator plots, the opacity here will change to show if it is selected.

Principle components view (F)

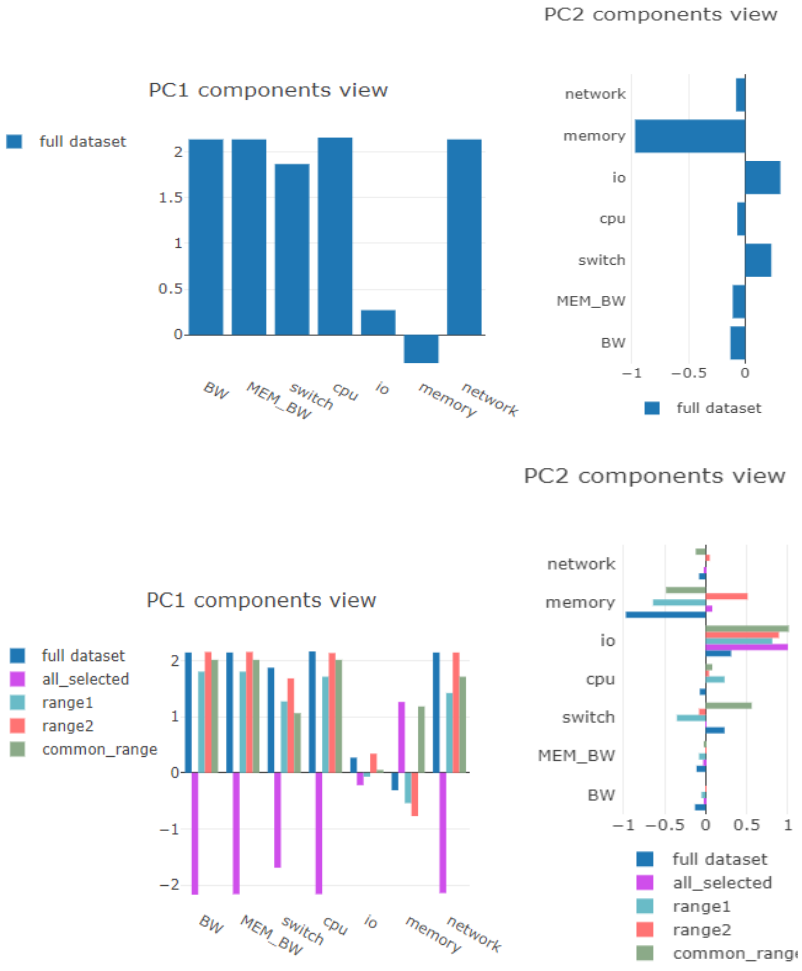


Figure 7 – Principle components view

- Lib
    - Plotly.js
- Data
    - PCA 2 principle components individual loadings(pca.components_) * Eigen values
- Encodings
    - X-axis(PC1 view): real attributes(system metrics)
    - Y-axis(PC1 view): ratio * Eigen values
    - X-axis(PC2 view): ratio * Eigen values
    - Y-axis(PC2 view): real attributes(system metrics)
    - Color: time period selected
- Interactions
    - Click on legend to select which range to show

From PCA we can get principle components which are a linear combination of data attributes. But how can human get the actual meaning of them, like how the principle components are related to the actual attributes? So I plot these two bar charts to show and compare the ratio of actual attributes count in principle component analysis result. To make them well scaled, I made them product by the corresponding eigen value. X-axis here is scaled ratio and y-axis is actual attributes. The direction of each bar indicates a position/ negative correlation. The longer the bar is, the larger the ratio is. To make comparison between selected time ranges and full dataset, I plot them as grouped bar chart. To make the plot looks simple and nice, I take advantage of the plotly library, user can determine which ranges to show by clicking on the legend. Comparing ratio of actual attributes allows people to observe the change in principle components better. And making ratio in uniform scale by multiplying them with corresponding eigen values makes correlation clearer.

Results

From the linking visualizations and interactions, we have achieved enough information to da analysis on the target application dataset. For this example, I take an application called ‘pmd’ to show an analysis case.

Findings

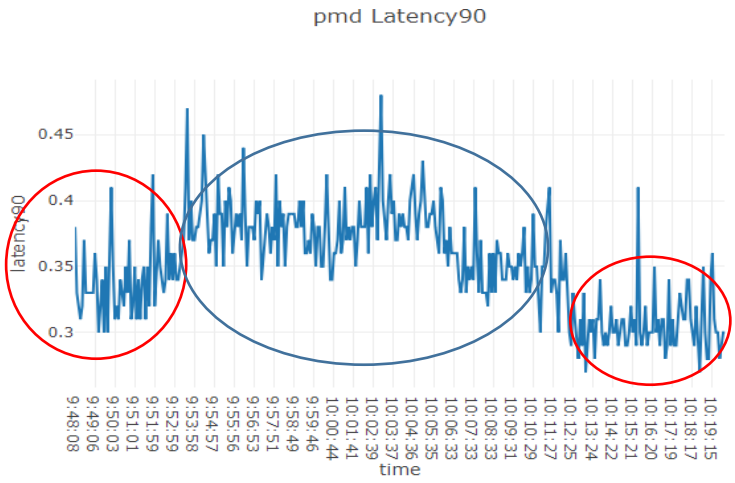


Figure 8 – Line chart

From Performance View, (View A), we can see at the beginning, ‘pmd’ shows fluctuation and from time 9:53 to 10:05, the performance stays steady. And from 10:06, latency tends to decrease until at 10:13, latency stays steady again at a lower value compared with previous time period.
What happened between these time periods, causing the latency up and down? Answering this question requires us to get an understanding in both the correlation of resource pressure and the variance in system metrics during these time periods. So let’s move on other plots to figure out why.

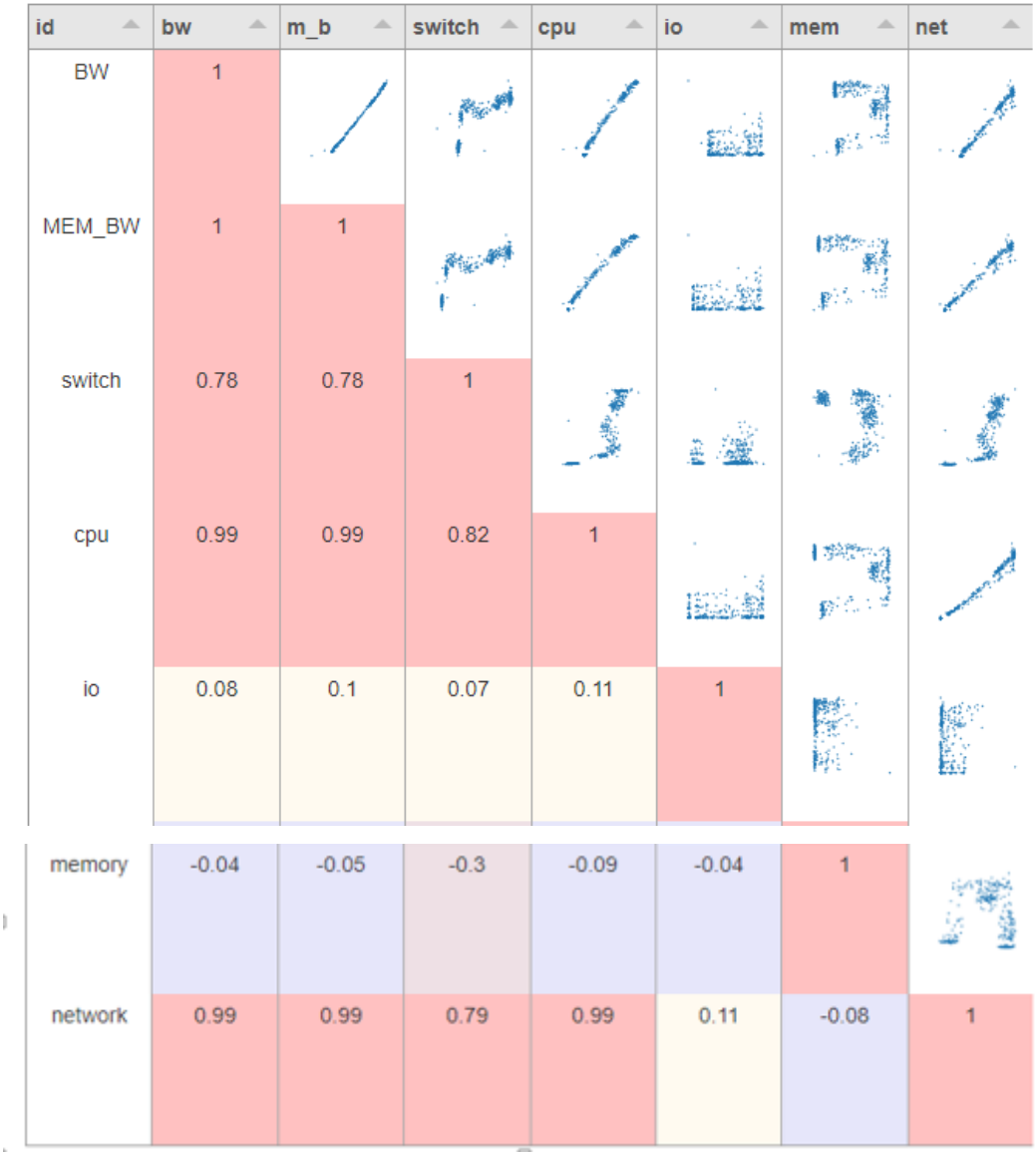
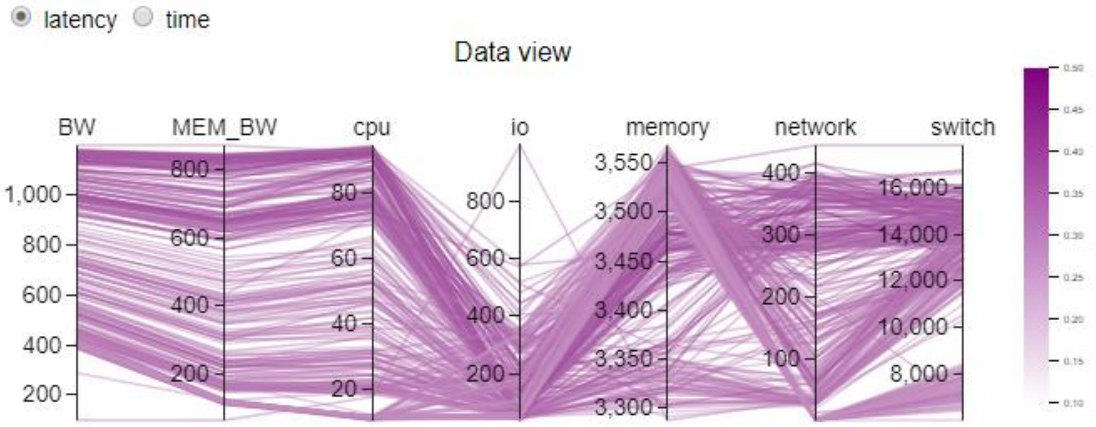
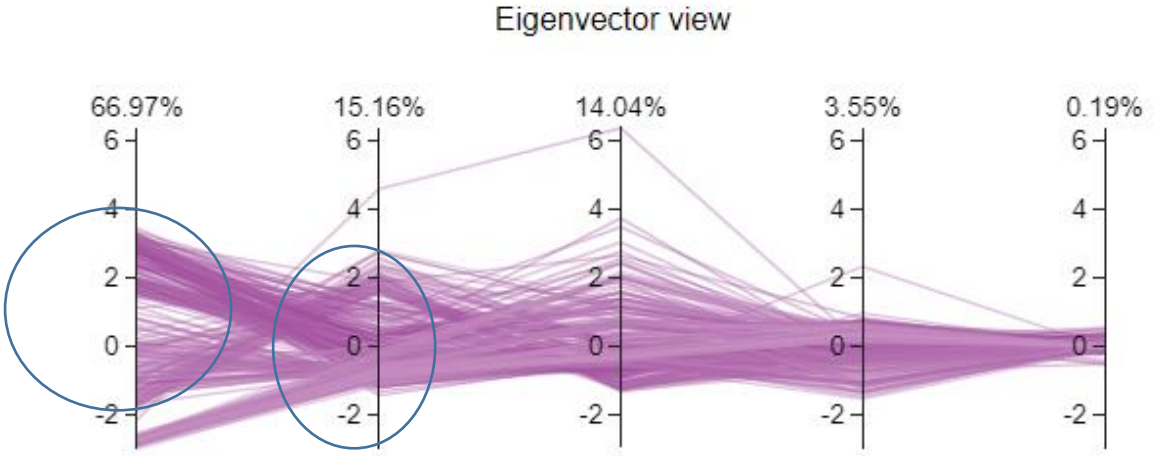
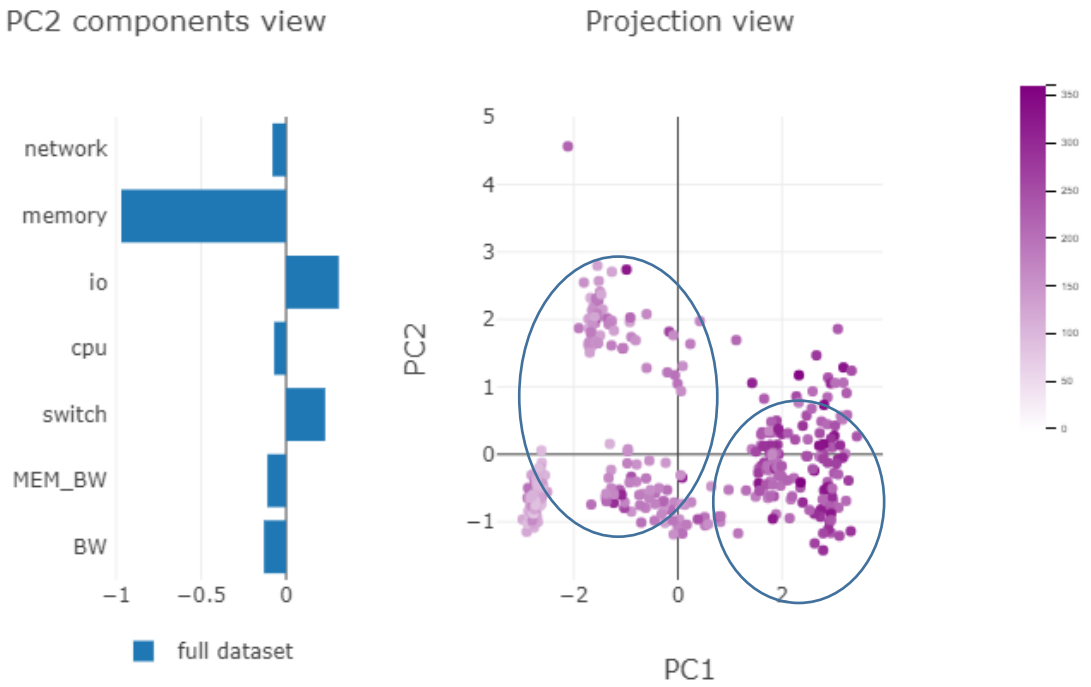


Figure 9

From Correlation view, (View B), for this full dataset we can see there should be a strong positive correlation between cpu percent and bandwidth, network and bandwidth, cpu percent and network. While there is negative correlation between context switch and memory.


latency
time
Data view
BW
MEM_BW
cpu
io
memory
network
switch


Eigenvector view
66.97%
15.16%
14.04%
3.55%
0.19%


PC2 components view
Projection view
network
memory
io
cpu
switch
MEM_BW
BW
full dataset
PC2
PC1

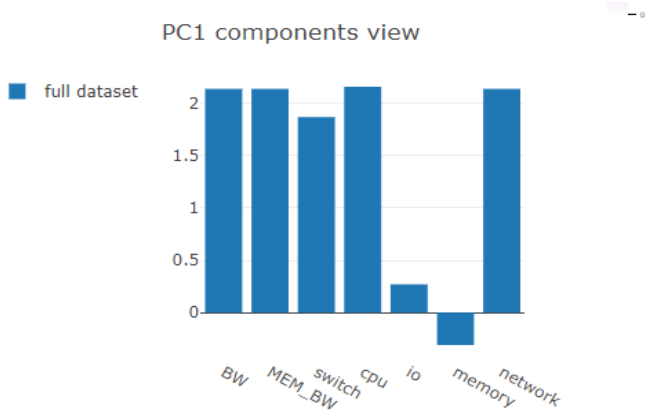
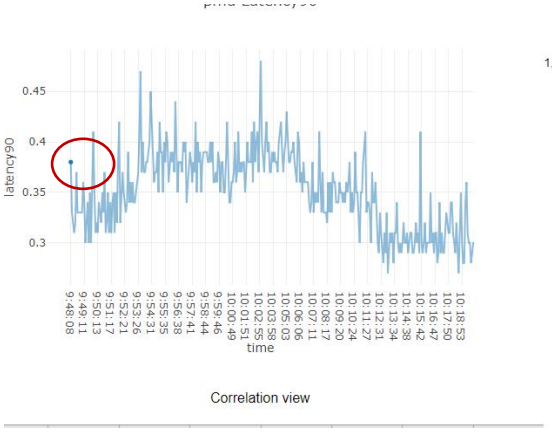


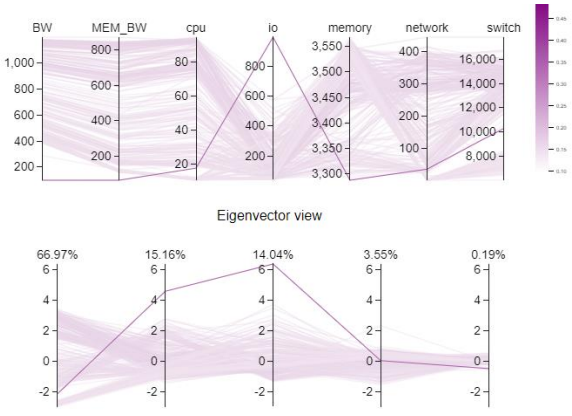
Figure 10

From Data View, Eigenvector View, Projection view, Principle Components View(View C, D, E, F), we can see, if the color encoding is assigned to latency, data view shows that, high latency record is more possible to have high bandwidth, memory bandwidth, high cpu percent and low io, low memory. If we look back to the correlation view, the correlation view shows the same thing. Then we look at the eigenvector view and projection view, they shows that, pc1 occupies 67% and data items with high latency are projected positive values at pc1. From components view that in pc1, the proportions of network, cpu, switch, memory bandwidth, system bandwidth are almost the same and they count for the majority of the components. This can also be inferred from the data view. Because most of data item seems to have high network, cpu, switch, memory bandwidth, system bandwidth values. Memory has a largest proportion in pc2 and you can see the data distribution in the second coordinator in eigenvector view appears a narrow range, and high memory value in data view. So all these plots conveys the same meanings but focus on different aspect to give the clear insight of data.

From the observations described above, we can get conclusion that, for the full dataset of 'pmd' application, cpu percent, bandwidth, switch, are strongly correlated. The higher these metrics' values are, the higher latency 'pmd' application gets. Higher memory bandwidth doesn't lead higher latency.

I found there's an obvious outlier in Data View, so I clicked on that line.





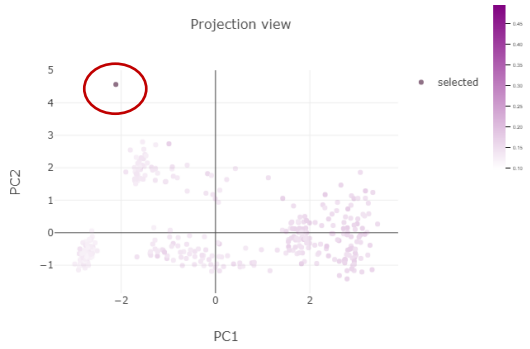


Figure 11

It shows that data item has very low value in each system metrics except io and has a high latency. The PCA projection shows it is also an isolated point. We can see this data item is actually the first data of the full dataset. And that really makes sense because the initial pattern for first few minutes could be potentially off, due to the uneven start of the target application.

Recall the three time periods in the line chart, now I know the reason of fluctuation at the beginning so we can ignore the initial time period. Let's do comparison on these two time period.

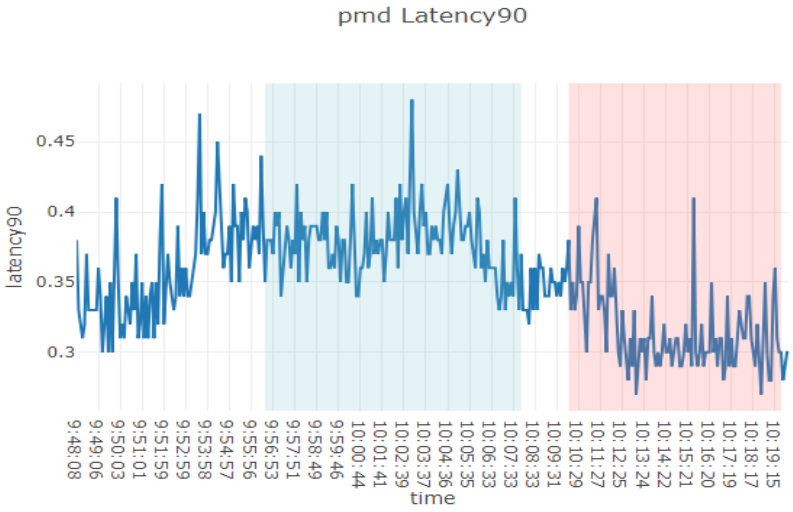


Figure 12

The first time period shows high latency and the lower latency.

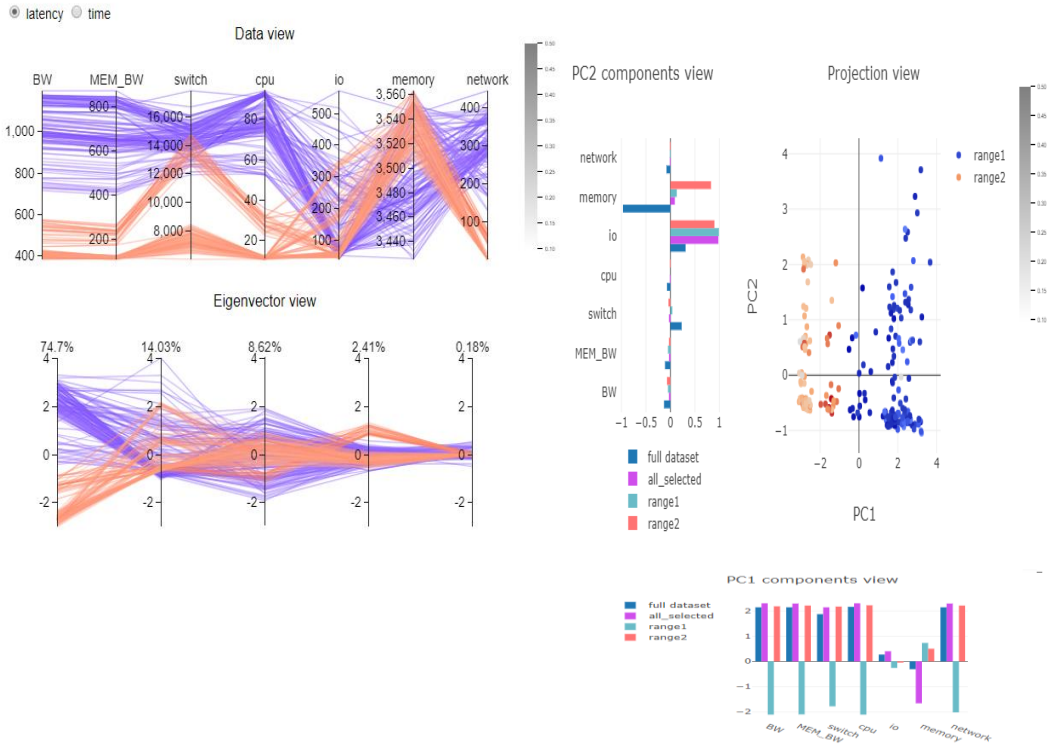


Figure 13

So I select the two time periods. The blue means the first time period with high latency and the red means time period the low latency. We can see the there is huge difference between the two time intervals in pc1 and actual attributes values. See the data view, High latency shows low memory, so maybe memory is a bottleneck for 'pmd' execution. And we can see the pc2 components view proves that.

Comparison with baseline

As what I mentioned above, the baseline paper 'iPCA: An Interactive System for PCA-based Visual Analytics' aims to assist the user in better understanding and utilizing PCA for analysis. It shows design prototypes for using parallel coordinates to view raw high dimensional data and corresponding eigenvectors, a scatter plot to project the dataset on the first two principle components, and provides bunch of interactive tools for a better understanding of the PCA model. I found the plot design of this paper really helps me. I took the idea and I added principle components view to show the relation between PCA results and actual attributes. To achieve my goal, I also added a line chart to show performance and provide users subset selection to make comparison. Also, the interaction of iPCA seems to aim at making the algorithm transparent. But for my proposal, my interactions are focus on how to lead users to get the analysis conclusions.

References

[1] FECBench: A Holistic Interference-aware Approach for Application Performance Modeling

[2] iPCA: An Interactive System for PCA-based Visual Analytics, Dong Hyun Jeong, Caroline Ziemkiewicz

[3] https://en.wikipedia.org/wiki/Principal_component_analysis

[4] https://en.wikipedia.org/wiki/Pearson_correlation_coefficient