An Efficient Framework to Derive Transaction Slicing with Performance Forecast for Enterprise Modernization

Ravikumar Ramadoss
Technology Architect
Infosys
Bangalore, India

Pushpa T
Department of Computer Applications
Alagappa University
Tamilnadu, India

N.M.Elango
Phd, Professor & Head,
Department of Computer Applications
RMK Engineering College
Chennai, India

ABSTRACT

During legacy application to Enterprise Modernization, where the Web logs are not available and deriving Transaction slicing for the 'n' number of applications to forecast the Performance is extremely difficult. This involves proper well defined process flows to derive the characteristics of heterogeneous applications. The performance forecast devised has to be approved and stamped by key stakeholders during the Non Functional requirements (NFR) sign off phase itself in any Enterprise Modernization [1]. This Paper emphasizes on the process flows to derive Transaction slicing with performance forecast for Enterprise Modernization.

General Terms

Transaction Slicing, Performance Forecast, Enterprise Modernization.

Keywords

Enterprise Modernization, Transaction Characteristics, NFR.

1. INTRODUCTION

Lots of Traditional applications are getting modernized to Enterprise applications. The key is to derive the performance forecast of the target modernized applications. As this forecast will be showcased to the key stakeholders during the enterprise modernization phase, the key characteristics of performance will act as the deciding factor. As majority of the applications have Transaction logs to derive the characteristics, the Transaction slicing can be derived using the available data points within the Traditional applications [2].

The purpose of this Transaction slicing is to categorize the transactions and forecast the performance of each transaction sliced. Based on the characteristics and forecast, the further Enterprise modernization can concentrate on how to effectively bring out the best results within the application by not losing its focus of key Nonfunctional requirements (NFR).

The objective of this paper is to provide an overview approach for Transaction slicing and an approach for the Performance forecast. This paper is organized as follows: Section 2 gives an overview of Transaction slicing. In section 3 gives a proposed framework for Transaction slicing and to derive performance forecast. In section 4 approaches and the API for transaction log parser are described. Section 5 and Section 6 consists of experimental analysis and Transaction slicing report analysis.

2. Transaction Slicing

Transaction slicing is the process of categorizing the transaction with simple, medium, complex and very complex category based on the characteristics. These characteristics are application specific like number of database hits, number of rules, and number of interfaces (externals). The primary source of information is the transaction logs and also from the business and technical SME, who preserves the existing business knowledge. The high level logical steps involved in Transaction Slicing are shown in below fig 1.

2.1 Preprocessing Phase

In this phase preserving of existing business knowledge will be utilized and key characteristics of the existing application are captured and stored in the master Meta data in the repository.

2.2 Pattern Discovery

In this phase pattern matching will be applied on the Transaction logs, and the characteristics are retrieved and stored in the desired repositories. The transaction log is the textual file where the existing application enters a record whenever transaction happens. Pattern and parsing the logs can help understanding the key attributes of the transactions.

2.3 Analysis and Forecast

In this phase the characteristics are analyzed and the key transactions are sliced against the SLA numbers and finally the report gets viewed using the bar chart.

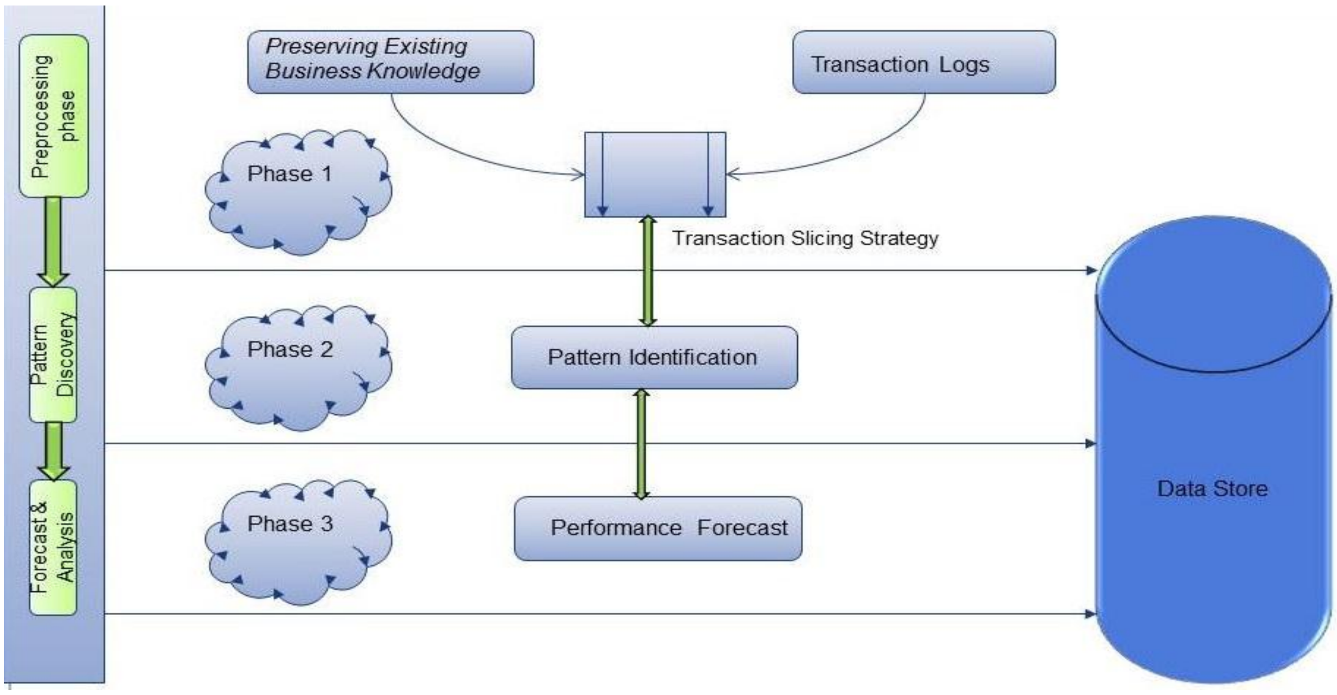


Fig 1: High Level Logical Steps Involved in Transaction Slicing

3. PROPOSED ARCHITECTURE FRAMEWORK FOR TRANSACTION SLICING

The high level proposed architecture is explained in the below fig 2. and the components involved in creating the Transaction Slicing are explained in the following sections.

3.1 Preserving Existing Business

In this component preserving of existing business knowledge will be utilized and key characteristics of the existing application are captured and stored in the master Meta data in the repository. The technical and business SME has huge amount of knowledge about how the existing application is interacting with in and also with other interfaces. This information is very vital in terms of coming up with high level Meta data structure.

Interaction with Technical, business SME: In this phase, interaction will happen between the modernizing technical team and the existing team.

3.2 Transaction log parser

In this component parsing of the transactional log happens and stores the Meta data with High level characteristics. The process needs to be repeated for ‘n’ number of application transaction logs available to identify the key application level characteristics and store the same into the repository. Efficient open source parsers will be used to filter out the logs.

3.3 Pattern Recognizer

This component uses a pattern matching technique that gets applied on the parsed Transaction logs, and all the matched patterns are retrieved and stored into the Repository. The characteristics are retrieved and stored in the desired repositories. The transaction log is the textual file where the existing application enters a record whenever transaction happens. Pattern identification and parsing the logs can help understanding the key attributes of the transactions.

3.4 Infrastructure component

This component is used for all the infrastructure related utilities like logging, security, auditing etc… and these will be applied in all the key components in the system. This is very important component as it provides all the value added services which are needed in other key component development of this Transaction slicing.

3.5 Performance forecaster

In this component, the parsed information from the Transaction logs are mapped against to the Key characteristics and then the performance forecast will be derived by the Technical SME team. Some amount of manual interpretation is needed in terms of review. Most of the forecasted data is automatically generated in the previous phases.

3.6 Report view

In this component, the stored details are retrieved from the database and get displayed in two views.

a) Graphical View: In this view, the retrieved data will be transformed into user friendly charts for various key transactions with in the system. This shows the stakeholders on how many transactions available and how the SLA will be on the modernized applications. It uses most of the components from the Infrastructure components phase.

b) Grid View: In this view, all the key elements of the transactional characteristics are retrieved and stored. It also uses most of the components from the Infrastructure components phase

3.7 Authentication

In this component, the details are only viewed by the authorized persons, who can view the respective authorized details. This component uses the enterprise sso, to build the plug and play authentication component into the system

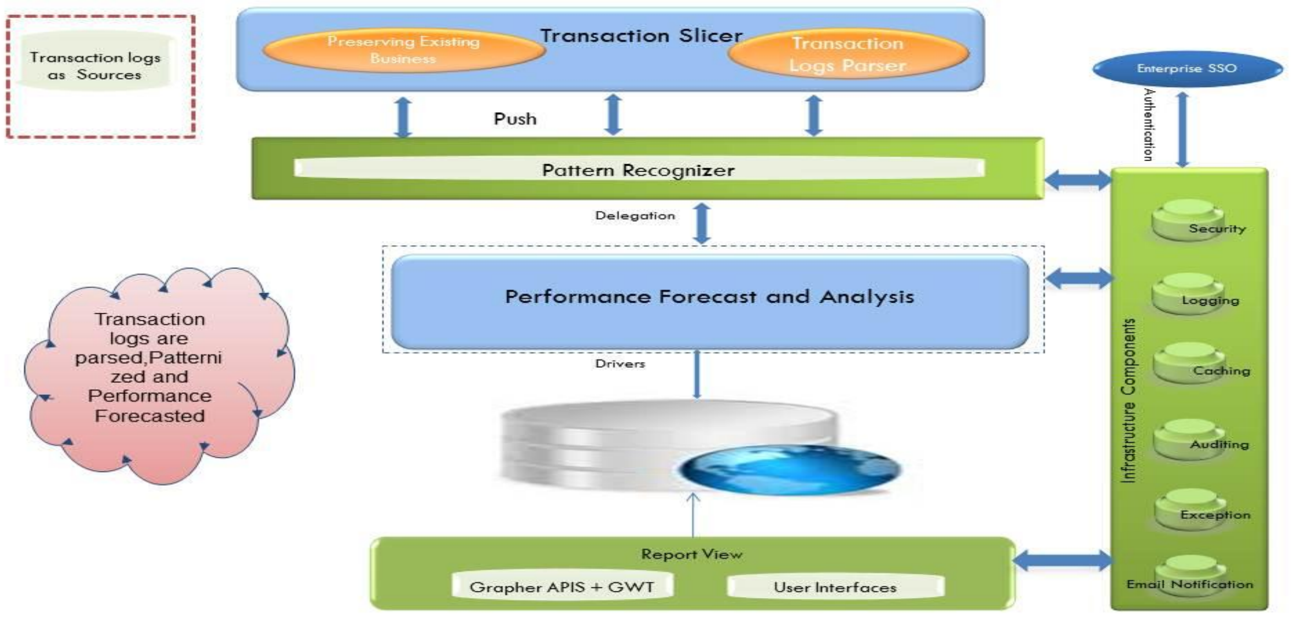


Fig 2: Transaction Slicing High Level Architecture

4. PROPOSED Work

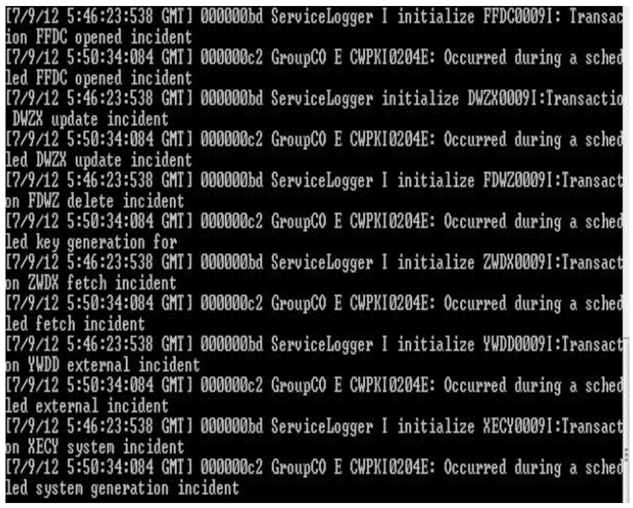
4.1 Transaction Log Parser

Several Transaction logs generated over a period of time contains various key transaction characteristics. This pattern recognized data will be derived from multiple source files which gets stored in a distributed way. These are captured and moved to the centralized repository before the parsing starts.

Parser technique is applied to read through the different Transaction logs and the retrieved information is passed on to the Pattern recognizer for further processing.

The Implementation of the component is done in Java and this component is built using the SAX Parser and Stream interfaces available in Java [3]. As this log files are huge in size, the parser is built in such a way that it can parse GB of log files in less than 2 seconds. Proactive performance engineering [4] has been applied on this component to ensure this component meets the required performance standards.

A portion of the transaction log file used for the experimentation has been shown in the following fig 3.



```
[7/9/12 5:46:23:538 GMT] 000000bd ServiceLogger I initialize FFDC0009I: Transac
ion FFDC opened incident
[7/9/12 5:50:34:084 GMT] 000000c2 GroupCO E CWPKI0204E: Occurred during a sched
led FFDC opened incident
[7/9/12 5:46:23:538 GMT] 000000bd ServiceLogger initialize DWZX0009I:Transactio
 DWZX update incident
[7/9/12 5:50:34:084 GMT] 000000c2 GroupCO E CWPKI0204E: Occurred during a sched
led DWZX update incident
[7/9/12 5:46:23:538 GMT] 000000bd ServiceLogger I initialize FDWZ0009I:Transact
on FDWZ delete incident
[7/9/12 5:50:34:084 GMT] 000000c2 GroupCO E CWPKI0204E: Occurred during a sched
led key generation for
[7/9/12 5:46:23:538 GMT] 000000bd ServiceLogger I initialize ZWDX0009I:Transact
on ZWDX fetch incident
[7/9/12 5:50:34:084 GMT] 000000c2 GroupCO E CWPKI0204E: Occurred during a sched
led fetch incident
[7/9/12 5:46:23:538 GMT] 000000bd ServiceLogger I initialize YWDD0009I:Transact
on YWDD external incident
[7/9/12 5:50:34:084 GMT] 000000c2 GroupCO E CWPKI0204E: Occurred during a sched
led external incident
[7/9/12 5:46:23:538 GMT] 000000bd ServiceLogger I initialize XECY0009I:Transact
on XECY system incident
[7/9/12 5:50:34:084 GMT] 000000c2 GroupCO E CWPKI0204E: Occurred during a sched
led system generation incident
```

Fig 3: Sample Transaction Log file

4.2 API DETAILS FOR TRANSACTION SLICING

The API and its interactions are explained in terms of class diagram and the same has been shown in the following fig 4.

These APIs are used in the process of Transaction Slicing to devise the required characteristics

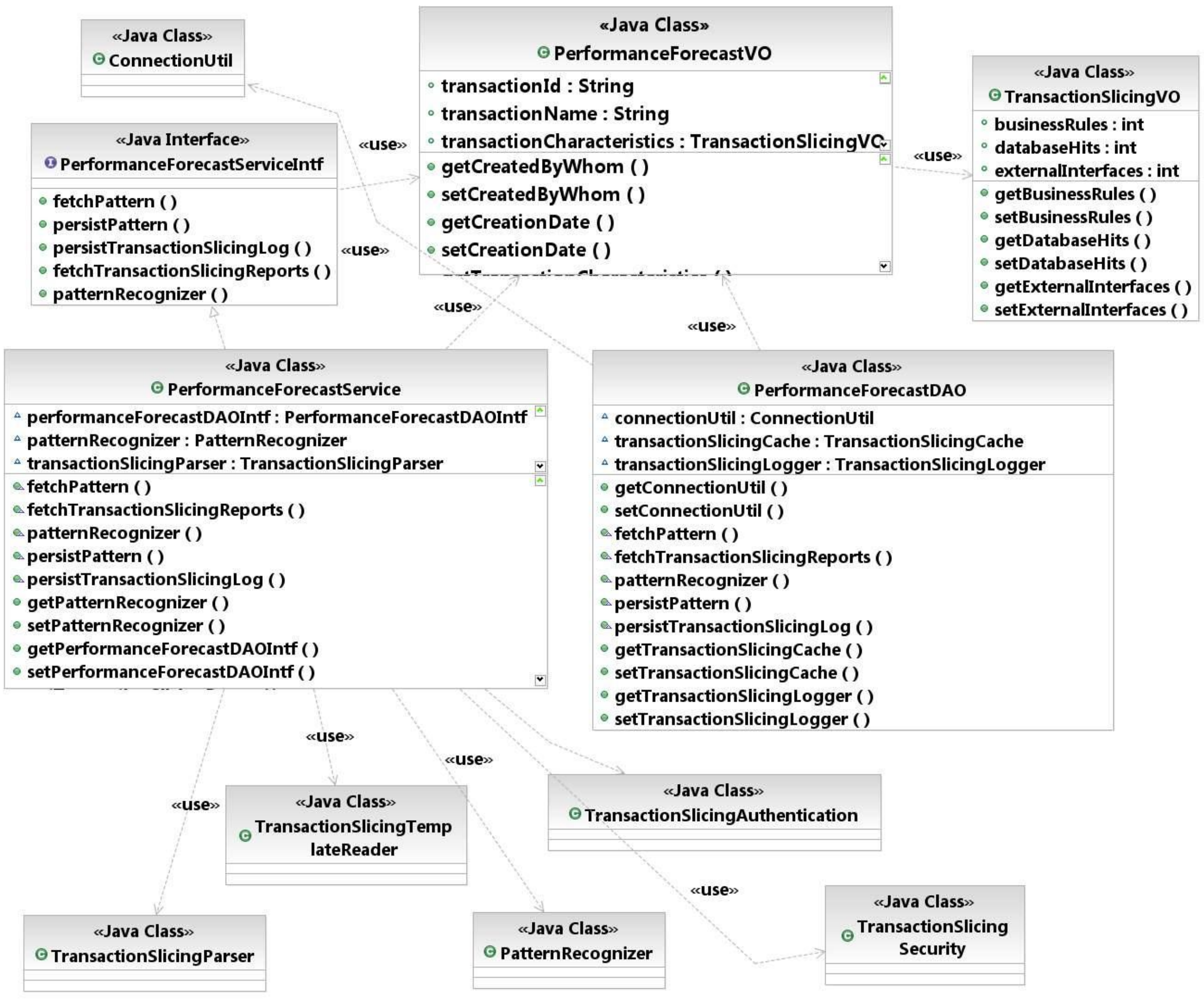

«Java Class» ConnectionUtil
«Java Class» PerformanceForecastVO
transactionId : String
transactionName : String
transactionCharacteristics : TransactionSlicingVO
getCreatedByWhom ( )
setCreatedByWhom ( )
getCreationDate ( )
setCreationDate ( )
«Java Class» TransactionSlicingVO
businessRules : int
databaseHits : int
externalInterfaces : int
getBusinessRules ( )
setBusinessRules ( )
getDatabaseHits ( )
setDatabaseHits ( )
getExternalInterfaces ( )
setExternalInterfaces ( )
«Java Interface» PerformanceForecastServiceIntf
fetchPattern ( )
persistPattern ( )
persistTransactionSlicingLog ( )
fetchTransactionSlicingReports ( )
patternRecognizer ( )
«use»
«Java Class» PerformanceForecastService
performanceForecastDAOIntf : PerformanceForecastDAOIntf
patternRecognizer : PatternRecognizer
transactionSlicingParser : TransactionSlicingParser
fetchPattern ( )
fetchTransactionSlicingReports ( )
patternRecognizer ( )
persistPattern ( )
persistTransactionSlicingLog ( )
getPatternRecognizer ( )
setPatternRecognizer ( )
getPerformanceForecastDAOIntf ( )
setPerformanceForecastDAOIntf ( )
«Java Class» PerformanceForecastDAO
connectionUtil : ConnectionUtil
transactionSlicingCache : TransactionSlicingCache
transactionSlicingLogger : TransactionSlicingLogger
getConnectionUtil ( )
setConnectionUtil ( )
fetchPattern ( )
fetchTransactionSlicingReports ( )
patternRecognizer ( )
persistPattern ( )
persistTransactionSlicingLog ( )
getTransactionSlicingCache ( )
setTransactionSlicingCache ( )
getTransactionSlicingLogger ( )
setTransactionSlicingLogger ( )
«Java Class» TransactionSlicingTemplateReader
«Java Class» TransactionSlicingAuthentication
«Java Class» TransactionSlicingParser
«Java Class» PatternRecognizer
«Java Class» TransactionSlicing Security

Fig 4: Transaction Slicing Class Diagram

4.3 HIGH LEVEL LOGICAL DETAILS FOR TRANSACTION SLICING FRAMEWORK

The following is the high level logical details for Transaction slicing

Table 1. Logical Details for Transaction Slicing.

Input: Transaction Log file

1. Trigger fetchPatterns API
2. Use the Connection Util to establish the connection to the Database
3. Read the Master Patterns from Database
4. Iterate around the Patterns and Store it in the Pattern Value Object
5. Close the Connection channel and return
6. Call parseLog API
7. Read the Transaction Logs one by one
8. Parse the Logs using TransactionSlicingParser utility
9. Utilize PatternRecognizer utility for the same
10. Filter the parsed logs based on the Patterns already stored in Step 4
11. Assemble and Aggregate the TransactionSlicing Value Object
12. Store the TransactionSlicingVO details on to the Database
13. Read Business Characteristics (Master data) from Performance Forecast table
14. Apply the Business Rules on the parsed logs
15. Persist the Business Rules filtered logs on to the Database
16. Iterate around the number of log files and repeat the same 6

17. Retrieve the Transaction slicing information from Database
18. Use Jasper Reports to print the retrieved records Graph View
19. Use normal Tag libraries to print the retrieved records in Grid ViewStore the Transaction Slicing details on to the Database

---

Output: Transaction Slicing Details with independent application specific characteristics

---

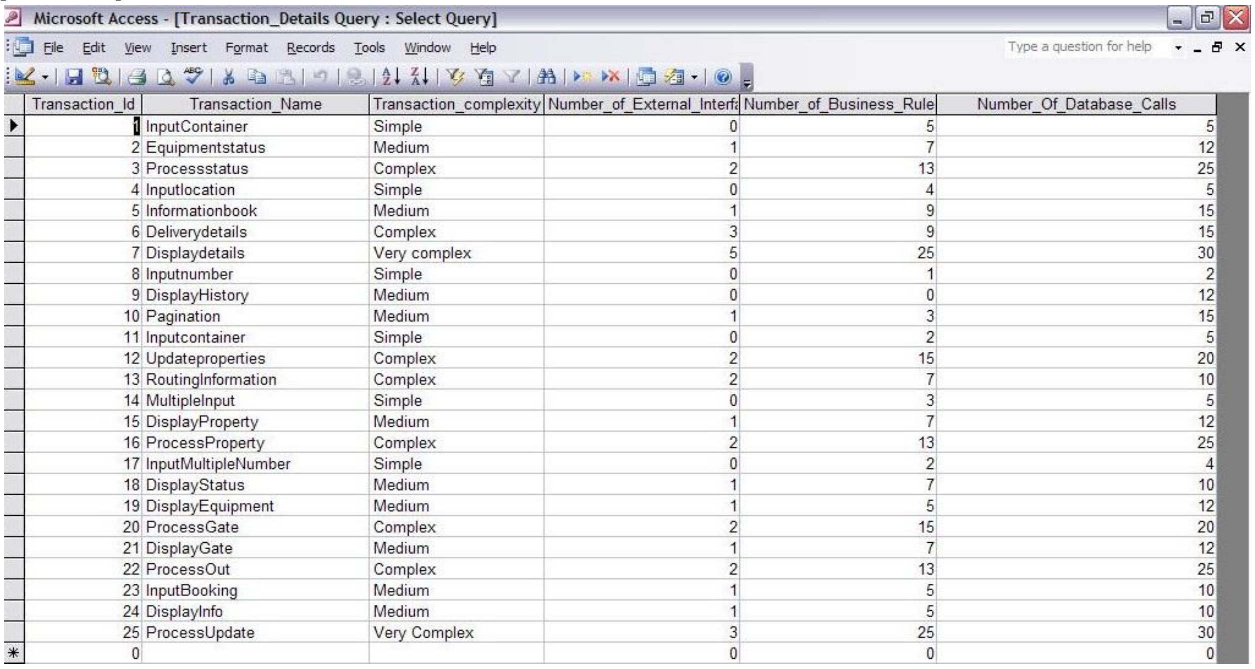
5. Experimental Results

The transaction log file used for this work got generated from 3 different legacy applications. The user arrival pattern is for 50 concurrent users using all three applications or one application to perform their transactions.

After the Transaction log has been parsed completely, the results are shown through the snapshots as shown in figure 5.

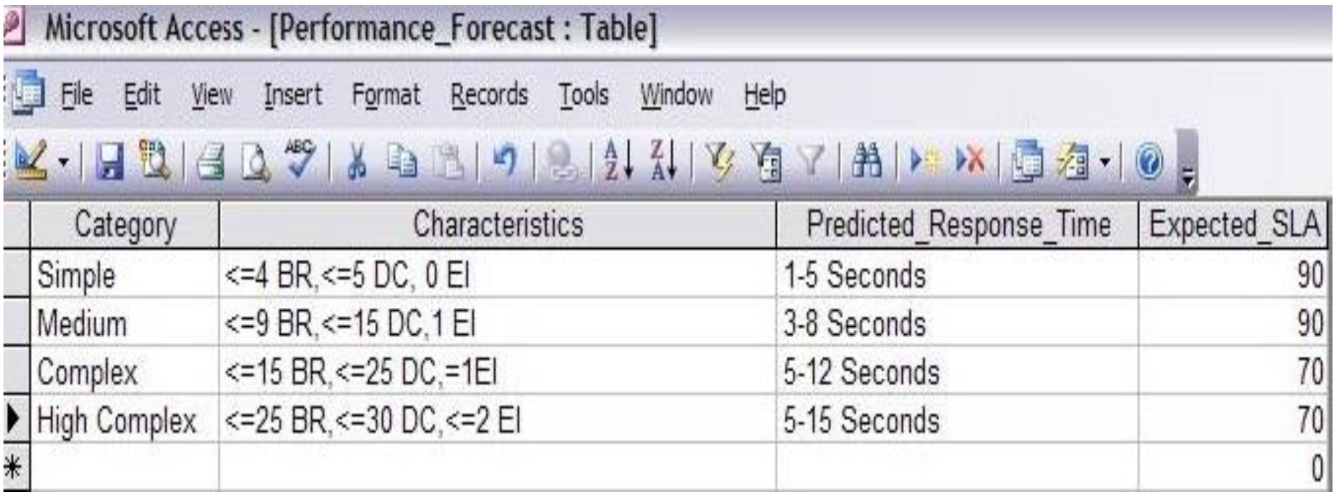
Figure 5 shows the results of the Transaction log parser. In this figure we can see that the different application key transaction characteristics are parsed by applying the pattern. The transactional slicing and its characteristics derived out are shown through the snapshots as shown in figure 6.

Figure 6 shows the results of the categorization and the key characteristics like number of database calls, number of business rules, and number of interface calls.BR represents Business Rules, DC represents database calls and EI represents Interfaces (external).


Microsoft Access - [Transaction_Details Query : Select Query]
File Edit View Insert Format Records Tools Window Help
Type a question for help

Transaction_Id	Transaction_Name	Transaction_complexity	Number_of_External_Interfa	Number_of_Business_Rule	Number_Of_Database_Calls
1	InputContainer	Simple	0	5	5
2	Equipmentstatus	Medium	1	7	12
3	Processstatus	Complex	2	13	25
4	Inputlocation	Simple	0	4	5
5	Informationbook	Medium	1	9	15
6	Deliverydetails	Complex	3	9	15
7	Displaydetails	Very complex	5	25	30
8	Inputnumber	Simple	0	1	2
9	DisplayHistory	Medium	0	0	12
10	Pagination	Medium	1	3	15
11	Inputcontainer	Simple	0	2	5
12	Updateproperties	Complex	2	15	20
13	RoutingInformation	Complex	2	7	10
14	MultipleInput	Simple	0	3	5
15	DisplayProperty	Medium	1	7	12
16	ProcessProperty	Complex	2	13	25
17	InputMultipleNumber	Simple	0	2	4
18	DisplayStatus	Medium	1	7	10
19	DisplayEquipment	Medium	1	5	12
20	ProcessGate	Complex	2	15	20
21	DisplayGate	Medium	1	7	12
22	ProcessOut	Complex	2	13	25
23	InputBooking	Medium	1	5	10
24	DisplayInfo	Medium	1	5	10
25	ProcessUpdate	Very Complex	3	25	30
0			0	0	0

Fig 5: Results of Transaction Log Parser


Microsoft Access - [Performance_Forecast : Table]
File Edit View Insert Format Records Tools Window Help

Category	Characteristics	Predicted_Response_Time	Expected_SLA
Simple	<=4 BR,<=5 DC, 0 EI	1-5 Seconds	90
Medium	<=9 BR,<=15 DC,1 EI	3-8 Seconds	90
Complex	<=15 BR,<=25 DC,=1EI	5-12 Seconds	70
High Complex	<=25 BR,<=30 DC,<=2 EI	5-15 Seconds	70
			0

Fig 6: Results of transactions categorization and its characteristics

6. TRANSACTION SLICING REPORT ANALYSIS

The above results show that the proposed methodology can be used to retrieve the Transaction key characteristics by using the Transaction logs and also more importantly the important application specific characteristics can be retrieved and showcased to the stakeholders for Signoff.

The high level characteristics which got captured during the transaction slicing by applying the business characteristics as shown in above fig 6.This clearly indicates by using the transaction slicing framework in Enterprise modernization considerable amount of Effort can be saved

The number of applications and the effort comparison by not using and using framework is shown in Table 1 and the graphical representation on the Effort saving in person hours in shown in Fig 7

The Expected percentile for meeting SLA's are represented by means of a graphical representation using bar chart as shown in figure 7

The different seconds of SLA has been sliced for simple, medium, complex and very complex transactions. The minimum predicted response time for Transaction is <=1 and the maximum predicted response time for Transaction is >=15

The result clearly indicates the amount of effort is directly proportional to the number of applications involved. If the complexity of the application is high and no Subject matter expert (SME) support is available, it will even increase the complexity to Slice the transaction and provide any forecast data points to the key stakeholders. Usage of this framework will not only reduce the effort involved, but also enables the key stake holders to gain confidence in Enterprise modernization.

The effort benchmark has been captured before and after the Transaction slicing and the key milestones are closely watched to do the comparison. Considerable effort gets saved when Framework is utilized.

Table 1. Comparison in Effort Saving Before and After Using Transaction Slicing Framework

Framework usage	No of Applications used	Effort in Person Hours	Effort Saving in %
Without framework	5	860	49%
With framework	5	425	

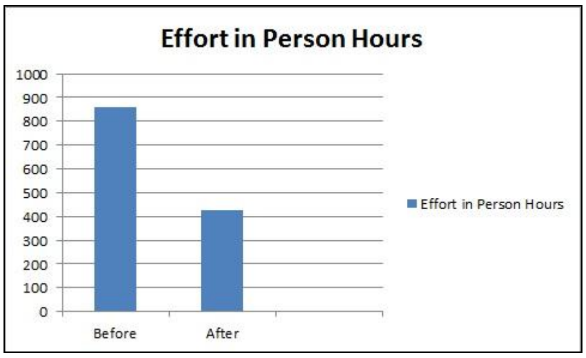


Fig 7: Effort saving before and after usage of Transaction slicing Framework

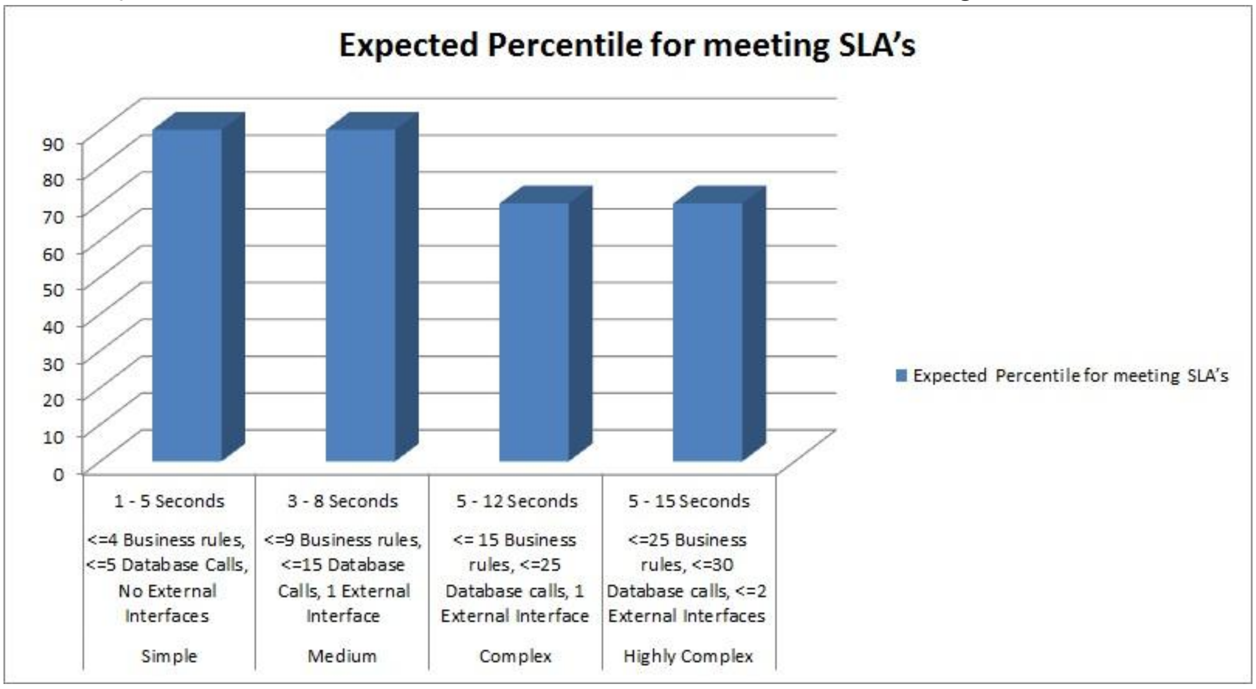


Fig 5: Results of Transaction Log Parser

7. CONCLUSION

Lots of legacy applications are getting transformed as Enterprise modernized applications and the key aspect is to understand the characteristics of the existing application.

When it gets modernized how much is the expected percentile for meeting SLA needs to be derived out to gain the confidence of the key stakeholders. The results shown are evaluated by the existing application SME team, and they found that the results are satisfactory and there are number of transactions where the SME itself not aware of as these applications are maintained for more than 6 years, and this Transaction slicing is helpful in bringing out the unknown important characteristics during Enterprise modernization. The existing framework is not SOA compliant and in future implementations this framework can be refactored to SOA standards and can be further made available as SAAS component for any organization or individuals to do efficient Transaction slicing. In the existing framework the master patterns defined in database is not extendable in nature. In future implementation this can be extendable by a Rich user interface where user can define the patters and it can be captured and persist into the database to enable user friendliness to efficiently slice the Transactions.

8. REFERENCES

[1] A Maturity Model for Application Performance Management Process Evolution. http://java.sys-con.com/node/1884172

[2] CrHa99 "Web Application Tuning, " Pat Crain and Craig Hanson, Proceedings of the 1999 Computer Measurement Group, Reno, Nevada, pp. 206-271, 1999

[3] LogDistiller log files merge sort tool, http://logdistiller.sourceforge.net/

[4] Performance Engineering and Enhancement. http://www.infosys.com/infosys-labs/publications/Documents/SETLabs-briefings-performance-engineering.pdf

[5] Build Better Applications from the start http://www.compuware.com/application-performance-management/dynatrace-development-team.html

[6] Patterns & practices Performance Testing Guidance for Web Applications. http://perftestingguide.codeplex.com

[7] The Business case for Software Performance Engineering. http://www.tarrani.net/mike/docs/BizCase4SWPerformanceEng.

[8] Software, Performance, or engineering? http://dl.acm.org/citation.cfm?id=584407

[9] Open source Java Reporting library. http://jasperforge.org/projects/jasperreports

[10] Proactive performance tuning for Enterprise Java developers. http://toadworld.com/Portals/0/ToadTechPapers/ProactivePerfTuningJProbeSQLOptimizer-US.pdf HMMT99 "ETE: A Customizable Approach to Measuring End-to-End Response Times and Their Components in Distributed Systems," Joseph L. Hellerstein, Mark Maccabee, W.Nathaniel Mills, and John J. Turek, International Conference on Distributed Computing Systems, 1999.