# PERFORMANCE COMPARISON ON JAVA TECHNOLOGIES - A PRACTICAL APPROACH

Priyanka Dutta, Vasudha Gupta, Sunit Rana

Centre for development of Advanced Computing,
C-56/1, Anusandhan Bhawan, Sector – 62, Noida
(priyankadutta@cdac.in, vasudhagupta@cdac.in, sunitrana@cdac.in)

***ABSTRACT***

*Performance responsiveness and scalability is a make-or-break quality for software. Nearly everyone runs into performance problems at one time or another. This paper discusses about performance issues faced during one of the project implemented in java technologies. The challenges faced during the life cycle of the project and the mitigation actions performed. It compares 3 java technologies and shows how improvements are made through statistical analysis in response time of the application. The paper concludes with result analysis.*

## 1. INTRODUCTION

Today's software development organizations are being asked to do more with less resource. In many cases, this means upgrading legacy applications to new web-based application with quick response time and throughput. Nearly everyone runs into performance problems at one time or another. Focusing on the architecture provides more and potentially greater options for performance tuning for improvement **[1]**.

In one of our project Pre Examination Process Automation System (PEPAS) we also faced such performance issues. A typical Examination System involves vide range functionalities dealing with public at large and an efficient communication and feedback mechanism to the utmost satisfaction of all the users. An error – free speedy interface is vital for successful functioning of the system **[2]**.There were many challenges while building the application. This paper tries to explore various Java technologies which could be adopted successfully for efficient application and performance improvement. Section 2 describes how the technology is chosen. Section 3 deals with various available technologies in Java, section 4 compares the performance of the technology used and conclusion.

## 2. TECHNOLOGY SELECTION

There were too many options available in terms of technology selection and keeping in view the requirements Decision Analysis and Resolution (DAR) one of the Project management techniques was utilized to decide on appropriate technology for the project. This Technique is intended to ensure that critical decisions are made in a scientific and systematic way. The DAR

David C. Wyld (Eds) : ICCSEA, SPPR, CSIA, WimoA - 2013
pp. 349–357, 2013. © CS & IT-CSCP 2013 DOI : 10.5121/csit.2013.3536

process is a formal method of evaluating key program decisions and proposed solutions to these issues [3]. Table- 1 depicts DAR sheet wherein how the technology was chosen to develop the system is shown.

| **Issues** | **Identified Alternatives** | **Evaluation Method** | **Evaluation Criteria** | **Remarks** | **Result** |
|---|---|---|---|---|---|
| 1.Presentation Tier Technology | JSP Java Applet Adobe Flex | ➢ Brainstorming<br>➢ Comparison | ➢ Ease of Building Interface<br>➢ Richness of User experience<br>➢ Ease of offloading Logics to client side without explicit installation of software at the client side Browser Independence. | ➢ Adobe Flex provides building Interface with easy to understand tutorials over JSP.<br>➢ Ease of Interaction with J2EE Middle Tier. (Adobe Flex) which is not possible in JSP. | Adobe Flex |
| | iReport Crystal Report | ➢ Brainstorming<br>➢ Comparison | ➢ Easy to build reports<br>➢ Rich features<br>➢ Open source | ➢ Ease of interaction with J2EE Middle Tier | iReport |
| 2.Technology for Middle Tier | EJB 2 Spring Grails | ➢ Brainstorming<br>➢ Comparison | ➢ Functionality<br>➢ Interoperability<br>➢ Advanced Technology | ➢ Use EJB 2 - less functionality over grails.<br>➢ Spring – Interoperability is complex.<br>➢ Grails – Advanced Technology Grails Groovy GFS (Grails Flex Scaffolding) with integration of Adobe Flex. | Grails |

TABLE 1: DAR Table made for PEPAS

After the careful evaluation of the technology the development bed was chosen. Grails and Adobe Flex were the chosen technologies as an outcome of the DAR.

Groovy is a dynamic language for the JVM that offers a flexible Java-like syntax that all Java developers can learn in matter of hours. Grails is an advanced and innovative web-application framework based on Groovy that enables a developer to establish fast development cycles through Agile Methodologies and to deliver quality product in reduced amount of time [4]. Grails Flex Scaffold (GFS) is a plug-in that deals with Flex code generation by scaffolding methodology [5]. When it came to attractive user interface the use of Adobe Flex was a powerful, open source application framework that allowed us to easily build the GUI of the application. [6]

The base model was developed and accepted by client. After acceptance the client placed different work orders. For every work order, customization of the base model was to be made as per the requirements. Timelines were set for each and every activity including requirements gathering, development and testing and a work plan was made at the beginning of the project. Constraints were highlighted. To control the project there is need to compare actual performance with planned performance and taking corrective action to get the desired outcome when there are significant differences. By monitoring and measuring progress regularly, identifying variances from plan, and taking corrective action if required, project control ensures that project objectives are met.

## 3. THE CASE ANALYSIS

Performance responsiveness and scalability is a make-or-break quality for software. Two work orders were completed on time, but following problems were persistent:

1. System Performance was slow
2. Stale Object Exception was thrown
3. Garbage Collector overhead limit exceeded

The problems were discussed and analyzed in the brainstorming session. Discussions were made for doing extensive code reviews. Using CAR (Causal Analysis and Resolution) technique for project management drawbacks in existing code reviews was identified and their corrective actions were planned. Revisiting and evaluating the design and architecture of the system once again were also discussed. The outcome of various code reviews were some changes like:

a) **Change in mail sending process** – Earlier mail was sent synchronously after the saving of registration data. Asynchronous Mechanisms of sending mail like installing and using Grails Asynchronous Mails Plug-in brings down the response time required for doing registration. Another advantage of using Grails Asynchronous Mails Plug-in is that the activity of sending mails may be scheduled or re-tried after certain amount of time. This feature would be useful in events when Mail Server is heavily loaded or Mail server is "down". The implementation of this plug-in requires minimal code changes in the current application.

b) **JVM options to tune JVM Heap size** - Java has a couple of settings that help control how much memory it uses:

   a. -Xmx sets the maximum memory heap size
   b. -Xms sets the minimum memory heap size

For a server with a 'small' amount of memory, we recommend that -Xms is kept as small as possible e.g. -Xms 16m. Some set this higher, but that can lead to issues e.g. the command that restarts tomcat runs a java process. That Java process picks up the same -Xms setting as the actual Tomcat process. So you will effectively be using two times -Xms when doing a restart. If you set -Xms too high, then you may run out of memory.

When setting the -Xmx setting you should consider a few things like -Xmx has to be enough for you to run your application. If it is set too low then you may get Java OutOfMemory exceptions (even when there is sufficient spare memory on the server). If you have 'spare' memory, then

increasing the -Xmx setting is often a good idea. Just note that the more you allocate to Java the less will be available to your database server or other applications and less for Linux to cache your disk reads.

Note that Java can end up using (a lot) more than the -Xmx value worth of memory, since it allocates extra/separate memory for the Java classes it uses. So the more classes are involved in your application the more memory that Java process will require.

The PermGen space is used for things that do not change (or change often) e.g. Java classes. So often large, complex applications will need lots of PermGen space. Similarly if you are doing frequent war/ear/jar deployments to running servers like Tomcat or JBoss you may need to issue a server restart after a few deployments or increase your PermGen space. To increase the PermGen space use something like: -XX:MaxPermSize=128m. The default is 64MB.

(Note that Xmx is separate from the PermGen space, so increasing Xmx will not help with the PermGen errors).

Since our technology selection was made using DAR hence we revisited our DAR and made two teams. One team worked upon Java/servelet technology and the second one worked on Spring Framework. For team one with java/servelet technology oracle database server has been choosen in place of mysql. A comparative study on query execution time was carried out and we found that query executed on MySQL is executing 30 times faster than the same query being executed on ORACLE. So we finally decided that for our application and for the kind of data which we record, MySQL database gives better performance.

## 4. PERFORMANCE COMPARISON

One of the most critical aspects of the quality of a software system is its performance and hence we set our goals to improve the performance of the system. Performance Test from Jmeter was undertaken. This was done iteratively after tuning the application and each time we get better result of performance from performance tests. Jmeter is one of the Java tools which are used to load testing client/server applications. It is used for testing the systems performance which automatically stimulates the number of users. It is also important to select the right kind of parameter based on your application for analyzing the test results. Since our application receives too many hits hence we choose response time as the evaluating parameter. Response time is the elapsed time from the moment when a given request is sent to the server until the moment when the last bit of information has returned to the client. We carried out the performance test on our base application developed with Grails and Flex. We gave inputs of 1 sec, 5sec, 10sec and 60 sec with sample sets ranging from 20 to 700 users and the report is as follows :

| Samples | Average (msec) | Min (msec) | Max (msec) | Std. Dev. | Error % | Throughput (/sec) | Avg. Bytes |
|---|---|---|---|---|---|---|---|
| **1 sec** | | | | | | | |
| 20 | 2722 | 1710 | 3472 | 499.36 | 0 | 4.88 | 327 |
| 100 | 31214 | 297 | 52071 | 19425.21 | 0.24 | 1.89 | 1357.64 |
| **5 sec** | | | | | | | |
| 20 | 315 | 260 | 505 | 64.47 | 0 | 3.97 | 327 |
| 50 | 3629 | 699 | 5387 | 1254.05 | 0.12 | 5.76 | 920.4 |
| 100 | 6098 | 638 | 11230 | 2818.44 | 0.43 | 7.95 | 2453.35 |
| **10 sec** | | | | | | | |
| 20 | 333 | 294 | 530 | 51.3 | 0 | 2.03 | 327 |
| 100 | 6177 | 320 | 9990 | 2431.46 | 0.04 | 5.98 | 524.8 |
| **60 sec** | | | | | | | |
| 300 | 494 | 267 | 3820 | 615 | 0 | 4.89 | 327 |
| 400 | 6248 | 333 | 15355 | 3947.63 | 0.1 | 6.03 | 833.86 |
| 500 | 37292 | 789 | 113203 | 30205.56 | 0.46 | 3.41 | 2195.23 |
| 700 | 68532 | 337 | 170815 | 43343.26 | 0.68 | 3.39 | 3088.47 |

TABLE 2: 1st Jmeter Report with Grails

From the above table it can be seen that average response time for sample set in 10 second for 20 users is coming to be 333 milliseconds with standard deviation of 51.3% which is very high and the throughput is 2.03 request /sec which is very low. Looking at these statistics it is evident that the system is unstable.

Performance Tests were also carried out for Java/servelet application which showed results as below:

| Samples | Average (msec) | Min (msec) | Max (msec) | Std. Dev. | Error % | Throughput (/sec) | Avg. Bytes |
|---|---|---|---|---|---|---|---|
| **1 sec** | | | | | | | |
| 1 | 2870 | 2870 | 2870 | 0 | 0 | 0.348432 | 51283 |
| 2 | 4296 | 3565 | 5027 | 731 | 0 | 0.361598 | 51283 |
| 5 | 8745 | 4510 | 12829 | 3063.947 | 0 | 0.36673 | 51283 |
| 10 | 15295 | 3597 | 26190 | 7458.177 | 0 | 0.369072 | 51283 |
| 20 | 27417 | 4175 | 50226 | 14277.99 | 0 | 0.390678 | 51283 |
| 50 | 65343 | 4730 | 125804 | 36155.33 | 0 | 0.394185 | 51283 |
| 100 | 96242 | 4240 | 129667 | 39525.4 | 0.49 | 0.764 | 26154.33 |
| **5 sec** | | | | | | | |
| 1 | 2795 | 2795 | 2795 | 0 | 0 | 0.357782 | 51283 |
| 2 | 3091 | 3016 | 3167 | 75.5 | 0 | 0.352734 | 51283 |
| 5 | 6767 | 4497 | 8903 | 1697.582 | 0 | 0.387447 | 51283 |
| 10 | 12644 | 3863 | 20973 | 5624.403 | 0 | 0.39248 | 51283 |
| 50 | 62592 | 4064 | 119374 | 34217.76 | 0.02 | 0.401858 | 50257.34 |
| 100 | 94558 | 4128 | 128009 | 39101.14 | 0.48 | 0.753551 | 26667.16 |
| **10 sec** | | | | | | | |
| 1 | 2734 | 2734 | 2734 | 0 | 0 | 0.365764 | 51283 |
| 2 | 2885 | 2787 | 2984 | 98.5 | 0 | 0.25047 | 51283 |
| 5 | 4424 | 3436 | 5284 | 692.505 | 0 | 0.384734 | 51283 |
| 10 | 10351 | 3982 | 17581 | 4326.702 | 0 | 0.393902 | 51283 |
| 20 | 23640 | 4082 | 44047 | 12011.69 | 0 | 0.386473 | 51283 |
| 50 | 61088 | 3460 | 117988 | 33411.58 | 0.02 | 0.397599 | 50257.34 |
| 100 | 59016 | 4325 | 100826 | 19939.48 | 0.57 | 0.91943 | 22051.69 |

TABLE 3: Jmeter Report with Java/Servlet

The above result shows that java/servelet version was no way near the grails version in view of performance of the application.

The Spring Framework provides integration with *Hibernate* in terms of resource management, DAO implementation support, and transaction strategies. When we did the performance test on this we found the following

| Samples | Average (msec) | Min (msec) | Max (msec) | Std. Dev. | Error % | Throughput (/sec) | Avg. Bytes |
|---|---|---|---|---|---|---|---|
| 1 sec | | | | | | | |
| 1 | 189 | 189 | 189 | 0 | 0 | 5.291005 | 102 |
| 2 | 191 | 191 | 191 | 0 | 0 | 2.828854 | 102 |
| 5 | 193 | 190 | 196 | 2.227106 | 0 | 4.911591 | 102 |
| 10 | 878 | 200 | 1243 | 273.9256 | 0 | 5.099439 | 102 |
| 20 | 1841 | 620 | 2997 | 753.7219 | 0 | 5.042864 | 102 |
| 50 | 5490 | 495 | 9441 | 2512.129 | 0 | 4.949515 | 102 |
| 100 | 11706 | 497 | 22748 | 6436.252 | 0 | 4.332568 | 102 |
| 5 sec | | | | | | | |
| 1 | 207 | 207 | 207 | 0 | 0 | 4.830918 | 102 |
| 2 | 218 | 217 | 219 | 1 | 0 | 0.732064 | 102 |
| 5 | 208 | 206 | 214 | 2.939388 | 0 | 1.188213 | 102 |
| 10 | 207 | 200 | 216 | 4.019950 | 0 | 2.118195 | 102 |
| 20 | 215 | 208 | 230 | 5.064336 | 0 | 4.031445 | 102 |
| 50 | 2811 | 280 | 6004 | 1486.374 | 0 | 4.936321 | 102 |
| 100 | 8869 | 311 | 17970 | 5450.867 | 0 | 4.577078 | 102 |
| 10 sec | | | | | | | |
| 1 | 219 | 219 | 219 | 0 | 0 | 4.56621 | 102 |
| 2 | 233 | 233 | 233 | 0 | 0 | 0.381025 | 102 |
| 5 | 227 | 224 | 229 | 1.854724 | 0 | 0.606796 | 102 |
| 10 | 225 | 220 | 230 | 2.98161 | 0 | 1.084599 | 102 |
| 20 | 223 | 219 | 235 | 4.130375 | 0 | 2.054232 | 102 |
| 50 | 888 | 331 | 2299 | 506.5207 | 0 | 4.483903 | 102 |
| 100 | 7162 | 373 | 17162 | 656.5207 | 0 | 4.445432 | 102 |

TABLE 4: Jmeter Report with Spring Framework

In the mean time when we were exploring and testing with other technologies we also did optimization of Grails version wherein we took following major corrective measures:

a) Removed bidirectional mappings which helped us to remove the unnecessary tables created by the mapping to store the values.
b) Explicitly clearing the Hibernate Session Level/1st Level Cache increases the performances. Hibernate 1st Level cache is a transaction-level cache of persistent data. It was seen that when this transaction-level cache is cleared, write performance of the system was increased. Initial load testing shows 5 x improvements in terms of Time Taken to insert for doing this test, 3000 inserts were made without clearing Hibernate 1st level cache which took around 240 sec. After clearing the Hibernate 1st Level cache, the same operation took 45sec.
c) By default Hibernate uses, Optimistic Locking strategies by the use of versioning. The system experiences high number of concurrent reads and writes. In such situations optimistic locking fails and throws an exception for Stale Object. The default behavior of Hibernate was changed from optimistic to pessimistic locking for avoiding this error. The change required

minimal code modification. The change however, may result in some performance penalty as there will be row level locking for reads.

d) For implementing row level locking, the Storage engine of MySQL was changed from MyISAM (which is tuned for High Performance without being ACID compliant) to InnoDB (which is tuned to support transactions and is fully ACID Complaint)

e) Worked on the query optimization and removal of unnecessary queries which were building load on mysql.

f) Initially the images were stored in the database along with the path and also on the application server which made the size of the application bulky and created lots of problems while retrieving the images for generation of application form or admit card. Now, the images are stored in a separate folder outside the webapps which helped in improving the performance of the application.

After doing all the above changes the application was tested with Jmeter to find the performance result which is depicted below:

| Samples | Average (msec) | Min (msec) | Max (msec) | Std. Dev. | Error % | Throughput (/sec) | Avg. Bytes |
|---|---|---|---|---|---|---|---|
| **1 sec** | | | | | | | |
| 1 | 47 | 47 | 47 | 0 | 0 | 21.2766 | 683 |
| 2 | 47 | 47 | 48 | 0.5 | 0 | 3.656307 | 683 |
| 5 | 48 | 47 | 52 | 1.939072 | 0 | 5.820722 | 683 |
| 10 | 49 | 46 | 54 | 3.257299 | 0 | 10.41667 | 683 |
| 20 | 50 | 46 | 59 | 4.093898 | 0 | 19.32367 | 683 |
| 50 | 791 | 146 | 1199 | 289.9754 | 0 | 23.57379 | 683 |
| 100 | 1973 | 104 | 3858 | 1051.758 | 0 | 23.94063 | 683 |
| **5 sec** | | | | | | | |
| 1 | 47 | 47 | 47 | 0 | 0 | 21.2766 | 683 |
| 2 | 48 | 47 | 50 | 1.5 | 0 | 0.784314 | 683 |
| 5 | 49 | 46 | 54 | 3.03315 | 0 | 1.230921 | 683 |
| 10 | 55 | 52 | 63 | 2.712932 | 0 | 2.194908 | 683 |
| 20 | 57 | 53 | 75 | 5.634714 | 0 | 4.148517 | 683 |
| 50 | 48 | 46 | 62 | 3.589763 | 0 | 10.006 | 683.38 |
| 100 | 51 | 46 | 62 | 4.430564 | 0 | 19.73165 | 683.19 |
| **10 sec** | | | | | | | |
| 1 | 48 | 48 | 48 | 0 | 0 | 20.83333 | 683 |
| 2 | 56 | 56 | 57 | 0.5 | 0 | 0.395491 | 683 |
| 5 | 54 | 51 | 60 | 3.544009 | 0 | 0.620887 | 683 |
| 10 | 58 | 52 | 79 | 8.971622 | 0 | 1.101443 | 683 |
| 20 | 56 | 50 | 76 | 7.031358 | 0 | 2.093145 | 683 |
| 50 | 49 | 46 | 68 | 4.422669 | 0 | 5.071508 | 683 |
| 100 | 49 | 46 | 77 | 4.272985 | 0 | 10.0311 | 683.19 |
| **60 sec** | | | | | | | |
| 300 | 53 | 47 | 106 | 6.930367 | 0 | 4.990435 | 683.19 |
| 400 | 51 | 47 | 114 | 5.434344 | 0 | 6.640327 | 683.19 |
| 500 | 50 | 46 | 96 | 5.398905 | 0 | 8.287201 | 683.19 |
| 700 | 50 | 45 | 178 | 8.207943 | 0 | 11.55726 | 683.19 |

TABLE 5: After optimization Jmeter report of Groovy Grails Version

The results showed that in 1sec with 1 user when compared with spring version Grails showed an improvement of 75% in response time and throughput increased to various folds from 5.2 requests/sec to 21.2 request/sec. A comparative analysis of the three technologies is shown below

with respect to response time and standard deviation. The results show that Grails is giving the best performance and hence is the rightly chosen technology.

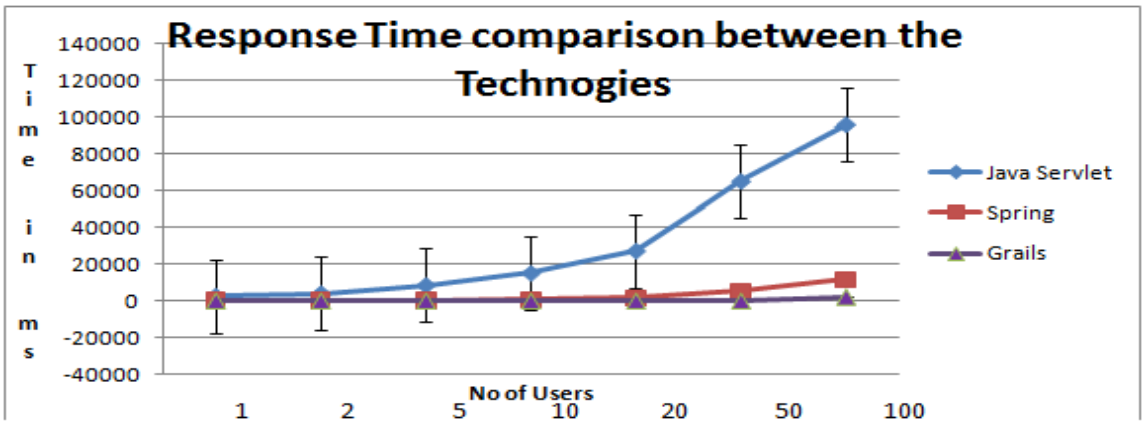


**Figure 1**: Graph showing comparative analysis of Table3, Table4 and Table 5 in average response time with number of users in 1 sec

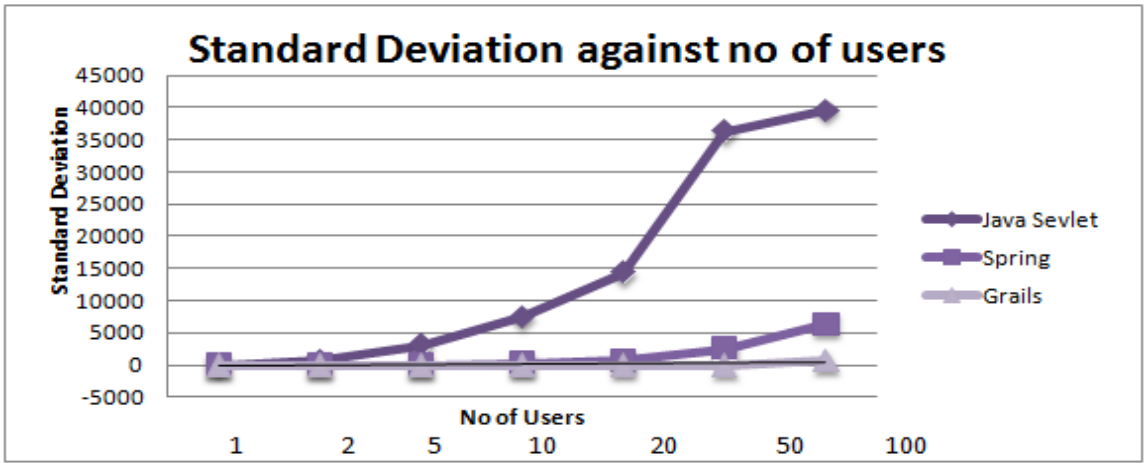


**Figure 2**: Graph showing comparative analysis of Table3, Table4 and Table 5 in standard deviation with number of users in 1 sec

When we compared this report to the first test report of Grails version we found that for 10 sec with 20 samples there was an overall improvement in response time and standard deviation. Response time is improved to 83% and there is 86% improvement in standard deviation which is remarkable in itself.

| Samples | Average( msec) | Min (msec) | Max (msec) | Std. Dev. | Error % | Throughput (/sec) | KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|
| 10 sec | | | | | | | | |
| 20 | 333 | 294 | 530 | 51.3 | 0 | 2.03 | 0.65 | 327 |
| 20 | 56 | 50 | 76 | 7.031358 | 0 | 2.093145 | 1.396111 | 683 |

TABLE 6: Comparison of 1st v/s Optimized Groovy Grails Report

It is also to be noted that now for 60 sec with 700 users the response time is 50 milliseconds and standard deviation is 8.2 and throughput is 11.5 request/sec.

## 5. CONCLUSION

Comparison of the three technologies namely Java/servelet, spring framework and Grails for performance led us to the result that Grails is a better performing platform for the project we undertook. Things like RSS feeds and domain modeling allows for faster development of the application while allowing the focus to be on functional code. The system through various optimizations has shown an overall improvement of 84 % in response time and 93% in standard deviations. In latest work order the system did not show any performance issues and the servers functioned smoothly without any downtime. After improvements done in our system we did not faced any memory leak issues. Through continuous improvement of the application we have been able to gain customer satisfaction.

## ACKNOWLEDGEMENT

The authors would like to thank Mrs. R.T.Sundari for her continuous guidance in writing the paper. They would also like to thank the EdCIL PEPAS team for their continuous effort in improving the application.

## REFERENCES

[1] Five Steps to Solving Software Performance Problems by Lloyd G. Williams, Ph.D.,Connie U. Smith, Ph.D.
[2] Application of the Decision Analysis Resolution and Quantitative Project Management Technique for Systems Developed Under Rapid Prototype Model by Priyanka Dutta, Vasudha Gupta, Santosh Singh Chauhan
[3] http://www.processgroup.com/pgpostoct05.pdf
[4] http://www.springsource.com/developer/grails
[5] http://grails.org/plugin/flex-scaffold+1
[6] http://www.adobe.com/products/flex.html
[7] http://grails.org/plugin/asynchronous-mail