International Journal of Computer Applications Technology and Research
Volume 6–Issue 1, 32-37, 2017, ISSN:-2319–8656

Performance Forecast of DB Disk Space

Srikanth Kumar Tippabhotla
CTS

Abstract: ln the absence of special purpose monitoring and/or modeling software designed specifically for forecasting database disk space requirements, a solution was developed using general purpose database facilities and office suite products. The results achieved were (1) an understanding of the heretofore unknown trends and patterns in the use of disk space by individual databases (2) the ability to accurately and proactively forecast the additional disk space needed for individual databases, and (3) the ability to reclaim the forecast unused disk space, all based upon linear regression analyses.

Keywords: Performance; DB

1. INTRODUCTION

As the sheer number, size, and complexity of databases deployed in organizations continues to climb each year, the effective management of those instances becomes a challenge for the IT staff, and in particular for those charged with maintaining database integrity, availability, performance, and recoverability. Practices rooted in reactive and frequently eleventh-hour individual heroics no longer make the grade in today's business environment. Rather, one needs to create and adopt repeatable, proactive methodologies in order to provide appropriate IT service delivery.
The database management systems (DBMS) utilized consisted of:

- Oracle Server (Oracle)
- Sybase Adaptive Sen/er Enterprise (Sybase)
- IBM DB2/UDB (UDB)
- Microsoft SQL Server (SQL Server)

Operating system (OS) environments included:

- UNIX -- IBM AIX
- UNIX -- Sun Microsystems Solaris
- Microsoft Windows Server.

As is the case with most operational support groups an on-call pager rotation schedule placed a team member in the "hot seat" 24 hours a day for a one- week period. During this tour of duty, the on-call person would respond to pages that were programmatically generated by the combination of the BMC Patrol Monitor for Sybase/Oracle/UDB product (hereafter referred to as Patrol) and the Tivoli Enterprise Manager suite. Additional, manually-generated pages were also issued by IT staff members at the round-the-clock computer operations center.

2. METHODS

Before getting into the details of the system, let's lay the foundation for that discussion with some background information.

Definition of Terms and Database Concepts:

DBMS: Software package that allows you to use a computer to create a database; add, change, and delete data in the database; sort the data in the database; retrieve data in the database; and create forms and reports using the data in the database.

Instance: A database instance consists of the running operating environment which allows users to access and use a database. A database (as a generic structured store of data) becomes an instance when instantiated as a system and made available via its database management system. Specific database providers can define database instances in terms of the precise hardware and software resources required to make them available: thus the Oracle database requires allocated system memory and at least one background process before the database counts as an instance.

Database: A database is a collection of information stored in a computer in a systematic way, such that a computer program can consult it to answer questions. The software used to manage and query a database is known as a database management system (DBMS). The properties of database systems are studied in information science.

in the case of Oracle and UDB there is a one-to-one relationship between an instance and a database; e.g. there is one and only one database associated with each instance.

Sybase and SQL Server, on the other hand, have a one-to-many relationship between an instance and its databases; e.g., one instance can have multiple databases defined and managed within it.

The Physical Representation of Database Content on Disk

All of the "stuff" that's stored and managed by a database (tables, indices, procedures, packages, rules, constraints) has to ultimately reside on disk. Below is an overview of the different approaches taken by the various DBMS architectures.

Oracle and UDB use the concept of a tablespace as the metaphor for holding the contents of a database. There is a one-to-many relationship between an instance (database) and its tablespaces. That is, an instance can and usually does have a number of tablespaces associated with it. Those tablespaces, however are not shared among other, unrelated instances.

A tablespace, in turn, consists of one or more "datafi|es" (Oracle) or "containers" (UDB) which are the actual physical files on disk that are visible to the hosting OS.

Sybase, on the other hand, use the concept of a database device to hold the contents of a database. A database device is somewhat akin to a tablespace. Database devices, in turn, consist of physical files on disk which are visible to the OS. While a database device can be used by multiple databases within an instance, the practice at the author's location is to

associate only one database to any particular database device. Therefore, for purposes of trending and analysis, disk

utilization metrics at the internal database level were gathered and used for forecasting, and not at the database device level.

ln order to define a common terminology for the tablespace (Oracle, UDB) and database (Sybase) constructs across the various DBMSs, the author coined the term “data holder”. When an instance or database experiences a near or complete shortage of disk space, it experiences that shortage at its “data

holder’ level. That condition is manifested in DBMS error messages to that effect. Similary, from the OS‘s perspective, the files which hold the databases content are stored just like any other file; i.e., inside an OS file system. While there are differences between the way the UNIX and Windows Server file systems work internally, logically they can be viewed as a pre-defined amount of disk space for holding files. Analogically, a data holder is to an RDBMS database as a file is to an OS file system -- both are layers of abstraction in the path to the final representation of database content on disk.

Regardless of how file systems are instantiated to a particular OS image (SAN, NAS, arrays, internal disk, others), they all share the common attribute of having been assigned a finite size that meets the anticipated needs of that file system. An image of an OS would typically have many file systems defined to it, each with a different size.

if a database instance has a 10GB file system defined for its use, that file system can hold any number of files as long as the sum of their sizes is <= 10GB. Any attempt to increase the size of an existing file or create a new file that would bring that sum over 10GB would be met with an OS error message and a denial of that attempt.

Statement of the Problem

No matter which DBMS is involved, all databases operate within the constraint of having to house all of their content within a set of data holders, each of which is pre-defined to be of a certain size. When any such data holder is first defined to the database, it will appear to the OS to be a file or set of files which occupies the full size of the defined data holder. For example, creating a 5GB tablespace in Oracle will result in a file or set of files whose sum of file system disk occupancy, as seen by its hosting OS, will be 5GB. However, from the DBMS's perspective at this point in time, the tablespace is empty or 0% full, and has no database content in it yet. It shows up as an empty tablespace with the capacity to hold 5GB worth of database objects. If the hosting OS file system were defined at 10GB, it would see the file system now as 50% full.

As database objects (tables, indices, etc.) are defined and subsequently populated using that data holder, it will present itself to the DBMS as housing n bytes. As n encroaches on 5GB it will come up against the 100% full internal DBMS mark and the DBMS will not be able to add any more content to that data holder until it's is made larger, or content is deleted. At that point the DBMS will return error codes to any

database operation which would result in the need for more disk space in the effected data holder; e.g., SQL INSERT requests and certain types of SQL UPDATE requests.) Note that there would be no OS error messages since no attempt has been made to increase the size of the underlying files.

Such a condition would be seen as a loss of availability to parts or all of the application using that database. In order to get the application running again, a short-term quick fix for this situation would be to increase the size of the data holder, providing that the hosting file system had unused space in it for such an increase. lf the file system were full, other IT players would have to be contacted to see if alternative solutions could be tried; e.g., the applications group might see if there was any data that could be deleted from the database, or the host system administration group would see if they could increase the size of the effected file systems. While these options were being explored, the application would be out-of-sen/ice. Were this to occur in the off-hours, an even greater delay in restoration to normal service would be expected as on-call people as contacted to remedy this basic disk space problem from remote locations. The magnitude of this issue became apparent to the author when he saw that that there were over 2,800 data holders that made up the Sybase, Oracle, and UDB instances. These 2,800 data holders in turn consist of over 5,100 individual data files. Managing 2,800 data holders and 5,100 files in a reactive, pager event-driven basis was simply not working. A proactive, quantitatively based forecasting approach was needed.

Statement of the Solution

ln order to prevent these types of database disk space problems from occurring, or at least to greatly reduce their likelihood of occurrence, what was needed was information that characterized the usage patterns of each of the data holders in the enterprise over time. Having that data would allow one to extract the underlying trends and patterns exhibited in the data holders over an extended period, and to forecast what the future needs were of each data holder.

To that end, the author devised the simple collector mentioned earlier for all of the Oracle databases. The collector itself is a single SELECT statement that leverages several PL/SQL features. This statement is run just once a day on a single Oracle instance. That single instance has database links set up for all of the other Oracle instances in the enterprise. This allows the SQL to gather the information from all other instances in the complex via database links, and to gather all of the information for all data holders on all instances into a single database table for analysis and longer term storage The PL/SQL loops through the dba_db_|inks table, generates the SQL needed for each instance, and then executes the generated SQL. The execution is serial, gathering the needed information about all tablespaces in any one instance and then going on to the next instance until information from all instances is placed into the central repository table. The information gathered from each instance was described above and is repeated here in its database format in Table below:

Column Name	Data type
Batch_date	DATE
Instance	VARCHAR2 255
RDBMS type	VARCHAR2 6
Data_holder_name	VARCHAR2 30
Allocated_bypes	NUMBER
Free_bytes	NUMBER

Note: “batch_date" is the date and time when the data was collected. ln order to provide a consistent value for all of the tablespaces in all of the instances at data collection time, the current date and time at the start of the collection process is stored as a constant. It's then reused in all of the data extracted from each instance, even though the actual extraction times might be a minute or two offset from that value. Given the

longitudinal nature of the data used in the analyses, this difference in time is not a problem. Having a consistent date and time for the all values collected that day allows grouping by date and time in subsequent analyses.

As noted earlier, shortly after the Oracle collectors were created and put into place, another team member created collectors for Sybase and UDB. These collectors gather the same six fields as shown in Table 1 since the data holder concept, by intent and design, is extensible to all DBMSs. Due to architectural dissimilarities between Oracle, Sybase, and UDB, the actual means of collecting the information is unique to each DBMS. The extracted data, however, has the same meaning and is not sensitive to any particular DBMS collection context. Once this information was gathered to cover a reasonable period of time, it was possible to subject the data and its derivatives to a number of analyses, described below.

Results:

Forecastinq Analvses Performed on the Data:

In order to use the collected data, it first had to be placed on a common platform that would provide the analytics needed for forecasting, descriptive statistics, etc. While the Oracle collector accumulated all of its obsen/ations about each Oracle instance into a single table on the central Oracle collector instance, the Sybase and UDB collectors used a different approach. They created individual flat files in comma separated value (CSV) format on each Sybase and UDB instance's host. What was needed was a means to gather the content of these three disparate data sources and put it in one spot. The initial solution chosen for this forecasting system was to use Microsoft's OLAP Services, a component of MS SQL Server versions 7.0 and 2000. The CSV files from each Sybase and UDB instance were automatically gathered together each day and sent via file transfer protocol (FTP) to an MS SQL Server instance. There they were loaded into a common table (t_data_hoIder) via Data Transformation Services. Similarly, the Oracle data for each day was automatically extracted out of its table and loaded into the same location. That table's layout is identical to the one shown in Table 1. All columns have the "NOT NULL" attribute. The intent of designing the data structure in this way was to provide a means of performing multi-dimensional analyses along variables of interest. While the main focus was to forecast when each individual data holder would run out of space, the presence of the instance, DBMS, and data holder columns allowed one to dice-and-slice the data along those values. These are discussed later in the paper.

The following derived measures were created for each data holder:

- Bytes_used: (bytes_allocated — bytes_free)
- Percent Used: (Bytes_used/bytes_allocated)*1O0
- Percent Free: (bytes_free/bytes_allocated)*100

With the table in place on MS SQL Server, the author defined a multi-dimensional data structure and set up analyses in MS OLAP Services. That structure and its analyses vetted (serial) time against bytes_used for each unique combination of instance name and data holder name. This analysis used the most recent 365 day's of daily data points for each data holder as input and yielded the slope, intercept, and Pearson product moment correlation coefficient (squared) for each data holder on each instance. Numerous other descriptive statistics were also calculated for each data holder such as the mean, median, mode, variance, standard deviation, and number of observations. The output of these OLAP Services analyses was in the form of a table which contained a row for each of the 2,800 data holders. The columns in each row of this table were the instance name, the DBMS type, the data holder name, slope, the intercept, the R2, the number of observations, the mean, the median, the mode, the variance, and the standard deviation for that "set".

Additional columns in this table, by way of programming done by the author within OLAP Sen/cies, were the most current values for bytes_used bytes_allocated, and bytes_free, along with the value of the date of the most recent observation in the past year's set of data for this instance/DBMS/data_holder set. Lastly, OLAP Sen/ices was further programmed to take these values and forecast what the expected shortfall or surplus would be, in bytes, for each data holder six months from the current date, using the method described immediately below:

Using the simple linear equation "y = mx + b", the author solved for "y" in order to see how many bytes would be in use (bytes_used) at time where "X" was displaced 180 days fon/vard from the current date. Applying the slope "m" calculated for each data holder we arrived at the projected bytes_used six months from now.

Further programming in OLAP Services yielded the difference between the most current bytes_a||ocated number and the six-month forecast bytes_used value If the difference between the current bytes_a||ocated and the forecast bytes_used was positive, that indicated that we would have a surplus of that exact magnitude six months from now for that particular data holder in that particular instance. lf, on the other hand, that result was negative, we would have a shortfall of that size in six months, and would need to take corrective action now so as to forestall an on-call coverage pager event in the future. The above calculations were all predicated upon filling the data holder to 100% of its allocated capacity at the six month target date, since we were forecasting bytes_used against the most current bytes_a||ocated. Based upon practical experience, and given the variance observed over time in each data holder on each instance, or collectively across the DBMS or instance dimensions, it was evident that allowing a data holder to approach 100% full was a dangerous practice. It did not take into account the periodic, seasonal, and random ebbs and flows that were observed in the data holder's behavior with respect to bytes_used. Not all data holders grew at a linear rate; rather, they exhibited troughs and crests in bytes_used over the course of time. The consensus within the author's workgroup, after having looked at the data for all instances and data holders, was that a best practice would be to forecast data holders to reach their 70% full "saturation" point. That being agreed, bytes_used was adjusted in the equation by dividing it by 0.70. The values of the surplus/shortfall column for all 2,800 data holders were then examined for the largest negative values. This was done by importing the OLAP cube into Excel. The conditional formatting feature in Excel was used to show those data holders with a projected shortfall to have their numbers literally "in the red". In some oases there was sufficient space in the underlying file system(s) to satisfy the forecast shortfall, and the data holder was increased in size by the amount calculated by one of the database staff. However, in other cases, there was not sufficient free space in the underlying file system(s) and a formal request was created for the UNIX disk management group to add the needed number of bytes. This proactive, forecasting approach allowed such

requests to be fulfilled well in advance of the six month projected shortfall. The forecast numbers, in and of themselves, were not used blindly. Examination of the R2 values for each data holder was used to assess how well the observed data points resonated to the march of time. If the R2 value was below 0.80, visual inspection of the plotted bytes used data points was undertaken to help understand the pattern, if any, in the data. If the data was “all over the place" for a particular data holder, the database team would make a best estimate of what to do with that individual data holder, and manually monitor it more closely in the coming six months. Since R2 is the percent of the observed variance that’s accounted for by the independent variable (time in this case), 0.80 was arbitrarily used as a line in the sand against which all forecasts were evaluated for usefulness.

These analyses were run on a regular and automatic basis every three months. The results were examined by the database group to see which data holders needed an adjustment to accommodate their projected growth (or decline) in the next six months.

3. BEYOND FORECASTING

3.1 Additional insights Provided by the data:

Having a year’s worth of data in the database now allowed the author to pose specific queries about the nature of all the databases in the organization. It was only by having this data and exploiting its emergent properties that these insights were possible Heretofore, such questions had been impossible to answer quantitatively due to there being no historical data. Best guesses were made, current real-time data was used, and the collective anecdotal experience of the workgroup was combined to produce SWAG answers.

Now, with the actual data at hand, a number of questions could be and were answered:

- Q. Which data holders exhibit has the highest or lowest mi of growth?

- A. By sorting the OLAP cube on the slope value, we display all data holders ordered by their rate of growth, from positive to negative.

- ' Q. Which data holders exhibit has the highest data content occupancy?

~ A. By sorting the OLAP cube on the mean bytes used value, we display all data holders ordered by amount of information they store.

- Q. Which data holders exhibit has the highest disk occupancy?

- A. By sorting the OLAP cube on the mean bytes_al|ocated value, we display all data holders ordered by amount of disk space they take up.

- Q. Which data holders are the most over- or under-utilized?

- A. By sorting the OLAP cube on their Percent Used or Percent Free values, we can display all data holders ordered by where they stand in the O-100% data holder full category.

Perhaps the most valuable insight was provided by

creating a pivot table/chart in Excel, which was published to the corporate intranet for use by managers, developers, business analysts, and others. This allowed IT staff to visualize the trends and other characteristics of the data in an interactive manner. Since Excel has an internal limit of 65K rows in a worksheet, and we had 2,800 data holders with 365 observations each, or 1,022,000 rows, the raw data could not be put into Excel. instead, the author elected to only use the data from the production instances, and to further aggregate that into its weekly mean values. This was done by writing a trivial SQL statement to find the weekly means for bytes_used, bytes_free, and bytes_a||ocated for each unique combination of instance name, DBMS type , data holder name, and week number within the year. (Since the data was on MS SQL Server, the datepart “week” value was used in the GROUP BY clause. The datepart "year" was used in the expression to order the data appropriately, since we had multiple years in the database.)

The data for the pivot tables and charts consisted of the elements shown in below Table:

dbms_prd_name (Sybase, Oracle, UDB)
db_instance (instance name)
data_holder_name
dbs_year (YYYY)
dbs_week (1-52)
Bytes_used (by that data holder)
Percent_Full (for that data holder)

Two pivot tables and two pivot charts were created from this data: one that showed the (absolute) bytes_used values, and one that showed the (relative) percent full values. The bytes_used pivot table and chart had the following characteristics:

- Its x-axis was time, expressed as the past 12 months, using the year and the week within the year. Since the past 12 months would span a year in all cases but the beginning of a new year, two pivot select buttons appear on that axis: one for year and one for week within year. For the most part these buttons were unused, and the entire 12 months of data was viewed.
- its y-axis plotted bytes_used.
- Pivot buttons were provided for:
  - dbms_prd_name
  - db instace
  - data holder name

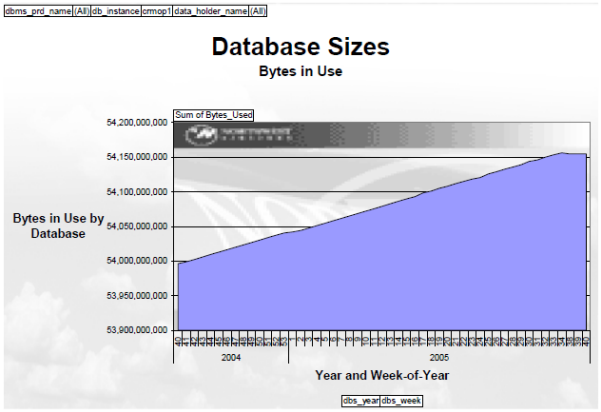
This allows the views to dice-and-slice the data along any of these dimensions. For example, these questions were answered in the pivot chart:

- Whats the pattern of bytes_used over the past year for:
  - All Oracle instances?
  - All Sybase instances?
  - All UBD instances?
  - Oracle and Sybase combined?
  - Oracle and UDB combined?

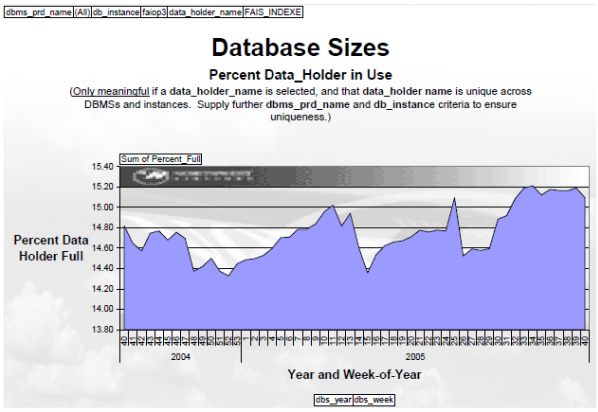
- Sybase and UDB combined?
- Sybase and Oracle and UDB combined?

- What's the pattern of bytes_used over the past year for:
  - Any individual instance?
  - Any combination of instances? (Note this also permits any combination of instances of interest. regardless of the DBMS that's hosting them.)
- What's the pattern of bytes used over the past year for:
  - Any individual data holder? (Note that one must enter an instance name for this to be meaningful. Otherwise it would show the total value for all data holders that have that name, regardless of the instance name.)
  - Any combination of data holders?

An example of this pivot chart is shows in Figure below:



The percent used pivot table and chart had the same setup as the bytes used pivot table. However, since the percent calculation was performed at the data holder level in the raw data, it would not be valid to do any roll-ups on the DBMS or instance dimensions. Therefore, a warning message was written to appear on the pivot chart that read Only meaningful if a data holder name is selected, and that data holder name is unique across DBMSs and instances. Supply further dbms_prd_name and db_instance criteria to ensure uniquenessm') Below Figure shows the pivot chart. By using this second pivot chart, people in the IS infrastructure could see which data holders are close to their 100% full limits. Also, the pivot table content can be copy/pasted into a new spreadsheet and then sorted on its percent full value to show all data holder's in the enterprise in order by their percent full values. Using excel Filtering, these queries can further be refined into DBMS type, instance name, and data holder name.



The Excel spreadsheet was created such that one can refresh the raw data from the source database on MS SQL Server with a single click. Therefore, each month it's possible to completely update the pivot tables and charts with virtually no effort.

4. DISCUSSION

By measuring and storing just these two, simple metrics every day for each data holder on each instance (bytes_allocated, bytes_free), the organization was able to evolve from its previous reactive mode to a more proactive and methodical process.

Benefits

Some of the benefits that accrued through this shift in focus and the use of applied mathematics were:

- With this data now published on a regular monthly basis to the intranet, the consumers of it have gained considerable insights into the seasonal and other variations in their data usage patterns.
- The work group responsible for acquiring disk space forthe entire IS organization can now set realistic budget values for next years disk space requirements, based upon the higher level rollups of the bytes_used data.
- Pager call reduction: the 1,041 pages that were previously issued per year for database disk space problems dropped to only a handful.
- The rates of growth of the various applications or business systems at the organization were now quantified and published. This allowed the IT organization to compare those rates between applications, year-over-year, etc.
- The organization can now identify any anomalousrates that might indicate that an application change (intended or not) or business driver variation was having a significant impact on the rate at which data was being accrued in a database.
- Descriptive statistics can be compared between data holders to better understand their central tendencies and dispersion characteristics.

5. CONCLUSION

lt's frequently amazing how just a tiny set of data observations can form the basis of uncovering the underlying substrates of an IT infrastructure. Automatically gathering just two values per day from each data holder in the enterprise allowed the IT staff to quantitatively and visually depict the patterns that had had been there all the time -- they had just not been measured. Applying that knowledge freed up considerable staff time which had previously been consumed by unnecessary, reactive paging events and by daily disk space monitoring. Now that these analyses have given us a glimpse into the nature of the organization's database environment, additional next steps can be considered:

- One could correlate (orjoin, in "database-eze") OS file system statistics with the DBMS data above so as to automatically determine if there is enough space in a file system to address the forecast deficit.
- We could explore non-linear regression relationships, any number of classical transforms (log, power, exponential, Nmorder polynomials, squares, cubes, inverses, etc.) of the dependent and independent variables could be performed to determine if any combination of those yields higher R values.

The very first DBMSs, which appeared in the early hunter-gather phase of IT, required a tremendous amount of staff time just to keep them running. Knowledge about and experience with them was scarce, often acquired as folklore from the tribal elders around the corporate campfires. Secret handshakes and magical amulets abounded. Mystical robes were frequently donned in order to exorcise the demons that plagued that software. However, over time, the DBMS vendors as a group added more and more self-managing capabilities to those systems, which made them less and less labor intensive. Even with those enhancements, the pesky problems associated with database disk space persisted. Some vendors did add features that self-managed the data holders by automatically extending them into their host file systems on an as needed basis, following business rules set up by the database administrators. Vendors are even putting in features that retract over-allocated data holders into disk space footprints that more closely resemble their normal usage patterns As those features become the accepted best practices in the workplace, the clerical tedium of managing database disk space will eventually become a faded memory, much like the punch card.

The system described in this paper is an attempt to provide a stepping stone to bridge the gap between the present state of DBMS capabilities and the future, and to do so with a methodology firmly rooted in quantitative analysis.

6. REFERENCES

[1] Vijay Datla, "Software Performance Tuning", IJARCST, vol. 4, issue 4, 2016.

[2] http://www.scolumbiasd.k12.pa.us/hsf/business/gengler/compapp/accintro.htm

[3] Vijay Datla "Performance of Ecommerce Implementation", International Journal of Advanced Research in Computer Science and Software Engineering, vol. 6, issue 12, 2016.

[4] http://en.wikipedia.org/wiki/Database_instance

[5] Vijay Datla "Software Performance Workload Modelling", International Journal of Computer Applications Technology and Research (IJCATR), Volume 6-Issue 1, 2017. doi:10.7753/IJCATR0601.1003

[6] http://en.wikipedia.org/wiki/Database_instance

[7] Vijay Datla "Performance Lifecycle in Banking Domain", International Journal of Computer Applications Technology and Research (IJCATR), Volume 6-Issue 1, 2017. doi:10.7753/IJCATR0601.1004