AR System Performance Improvement Methodology

By Nicky Madjarov

Contact information: Nicky Madjarov
email: nicky@speedupars.com
phone: 973-202-4278

Version 1.1
March 2009

Check for the latest version of this document and other related articles at http://www.speedupars.com/

Introduction

This methodology is developed for those of you who utilize enterprise-wide BMC Remedy ITSM solutions and other Action Request System based custom applications, and are facing poor performance problems. The audience could vary from Remedy developers, dba's, to ITSM service managers.
Because the technology is constantly changing, this methodology will be systematically updated. The latest version, as well as other related articles, could be found at www.SpeedUpARS.com.

Why methodology?

There is virtually no single tool that can address the complexity of application performance. Yet we have to address the performance problems in systematic and timely manner.
Even with the entire variety of environments and applications that could be in place in particular implementation it is still possible to follow a systematic approach in identifying performance problems and applying solutions for performance increase.
This methodology presents a multi-phase approach for identifying and resolving your performance problems.
Ideally, if you follow the ITIL methodology, you'd already have recorded evidence about performance issues in the form of incident reports and problem records. If the problems have been carefully analyzed and resolved in timely manner you probably would not be entering this performance improvement process. But let's just assume that you are new to the performance problems and they just recently surfaced in your environment.
This methodology is a combination of administrative and technical actions that, when executed properly, it will improve your application performance significantly.

1. STEP 1: ACKNOWLEDGE THE PERFORMANCE PROBLEM.

At some point someone has to say "Huston, we have a problem ...". It has to be said loudly enough in order to stress its importance. Do not avoid confronting the problem by "We have a ticket open with the vendor ..." or move the ball somewhere else - "It was assigned to XYZ group to investigate". Create a problem entry in your Problem Management application, or if you don't have one, just follow the process established in your organization for problem resolution. Either way, everyone, including your management, have to be aware of it.

Perform the initial due-diligence. Investigate the history of the problem, old (resolved) problems that may be related to this one, applied solutions, resolved incidents, open incidents, identify the user population that is affected by the problem.

Now, why is this important, isn't it more convenient to keep it quiet and hope that one day the problem will disappear. Well, the problem resolution may require human and financial resources. By acknowledging the problem you prepare your way for the next steps in the problem resolution process.

2. STEP 2: REVIEW KNOWN SOLUTIONS.

There are quite a few known application defects or weak spots that cause performance loss. Most of them have solutions or workarounds. Some of them may still be open with the vendor. Prepare a list of them and find out if the available solutions or workarounds have been already applied in your environment. If they are not, take the necessary steps to implement them and get your environment up to date.

Do not close your problem record. Re-access the performance problems. If you have not achieved satisfactory results, proceed to the next step.

Keep the un-resolved issues on your list, this may help you to prevent re-discovering them.

3. STEP 3: FORM A TEAM

Working in complex environments provides for vast variety of performance loss possibilities. The performance loss is almost always sum of performance losses created by different causes. Given the vast variety of administration and diagnostic tools available for the different service components and the restricted access to each of them require to put together a team that covers them all. Do not make preliminary assumptions about the root causes and cut off some team members that may not be necessary. Ask the fellow managers to identify resources from their teams to work on this problem. On a very generic level you should have covered the following:

- Remedy Administrator (application architect if available)
- OS Administrator for Remedy application server(s)
- OS Administrator for Mid-Tier environment
- OS Administrator for database server
- DBA for the Remedy underlying database and external databases if used
- Application Server (Web Server) Administrator for Mid-Tier
- Reporting Server Administrator
- Administrator(s) for integrated solutions - LDAP, SSO, etc.
- Representative from Network Management team

Once the team is put together, hold a kick-off meeting to introduce everyone and present the findings.

4. STEP 4: ANALYZE THE CIRCUMSTANCES OF PROBLEM OCCURRENCE

Whether it is already recorded in the form of incidents and problems or not, you should create a table describing the exact circumstances when the problem occurs. In a very extreme case you may end with one item like "whatever action the user is trying to perform it takes forever". Don't laugh, it is important to know it.

The goal of this analysis is to find patterns that will help you concentrate on narrower areas in your further analysis.

You may consider grouping your findings as following:

- By location - problem is specific for one or more locations
- By time - problem occurs in specific timeframe
- By operation - problem occurs when performing specific application functions
- By client type - Web, User Tool, different OS's

If you don't have found any pattern, this is very significant finding by itself.

5. STEP 5: PIN-POINT THE ROOT CAUSES

Please, work with the assumption that the poor performance is result of multiple causes and do not abandon the process when you find one of big significance.

Look at the patterns that you found in the previous step.

5.1. GLOBAL PERFORMANCE PROBLEMS

If you have global performance loss (e.g. everything the user does takes forever) do the following:

- Analyze the application server and database server health. Monitor them both to make sure that they have available memory and CPU resources at all times. If your server is paging and the CPU utilization is too high you must resolve this problem before going any further.
- If your servers' health is good you need to check your network response time. You can use simple utilities like ping and tracert to get an idea about the time spent for communication between clients and servers. If you suspect that this process takes too long, ask your Network Management for further help. Work with the network Management to establish if the bandwidth between different application components is sufficient. Make sure that there are no DNS resolution issues.
- Check if the list and fast threads are sufficient to serve the users in regular (peak) hours. Such information can be obtained from the arhtread log or by analyzing the statistics generated for AR server (Server Statistics form). Keep in mind that the latter contains accumulative values. If this analysis shows that user requests is waiting for a list or fast threads to become available, increase the initial and maximum number of server threads and keep increasing until you achieve free server threads of each type available at all times.

Note: each server thread will use server resources, in particular memory and CPU, do not blindly increase these numbers because this can decrease your overall application performance.

Note: You should have significantly more list servers than fast servers. There are many "rules" out there how to determine the appropriate number. I can only share my experience: ratio 1:5 fast:list has worked very well for me. After every increase check the memory and CPU utilization of the application server.

- Check for logging running routinely all the times. Pay special attention to logs generated on the servers: filter, escalation, api, sql, user, mid-tear, database, os, etc. Of course, you are encouraged to generate logs and analyze them, however, once you accumulate sufficient information for your analysis, just turn them off.

Note: Some users (especially in the pharmaceutical and financial industries) has been told in the past that recording constant logs and archiving them will make them compliant with certain government regulations. This is not true. If this is your case and you are doing it because someone advised you so, find another regulatory specialist to help you out and stop doing it.

- If your environment is using external user authentication or single sign on you should perform proper analysis on the service you use. Please, note, that every API calls performs user authentication and possible delay will impact your overall application performance. Each action you take is translated to API calls.

All these actions are quick are inexpensive. They are performed by your own resources. It is recommended that you do check them regardless the patterns you may have found.

5.2. SINGLE ACTION PERFORMANCE PROBLEMS

- Perform interviews with users, review existing incident/problem reports. That can expand and clarify the information you have collected in Step 4. Some patterns could become clear.

If you have generated server statistics you will have good starting point for your analysis. Generating combined logs of api/sql/filter/active link may help you establish where delays occur, then you can relate to specific object, and finally, associate with application function. These will point you in the direction of the problems.

- Ask your dba to provide top SQL list. Use logs to find out what triggers the top SQL's. Update the table you created in Step 4.

Here is the milestone where the performance improvement process loops. If you have checked all of the above, applied fixes, and you still have global performance issues, you need to take a closer look of what you have done and possibly re-do it.

If you have discovered patterns on location, operation, time, client, etc., you should start analyzing them in greater details one by one. There is no particular order to follow, I will order them in what, in my opinion, is the easiest to perform.

5.3. Pattern Analysis

5.3.1. Location patterns

If you have established that certain group of problems is related to one or more locations, concentrate on these locations. Create logs from these locations and analyze them. Work with your Network Management to analyze the traffic, DNS resolution, and other location specific circumstances. Utilities like tracert and nslookup can give you some initial idea about network and DNS response times. If you establish that the problem is related to insufficient of bandwidth create a problem record, identify the locations affected, document the findings of your root cause analysis. Even if you cannot resolve the problem in the near future, the existence of such a problem record is important for future reference and planning. Move your further analysis away from these locations. Even if there are more problems to be found your better choice would be locations that don't have location specific performance problems (you will avoid the "noise" of a known problem).

5.3.2. Client patterns

If there are performance problems related to specific versions or patches, separate them in problem classes and use alternative versions and patches to prove that the problem is really related to one of them. If you prove it, replace the current ones with the ones that work better.

Note: Web based clients usually work slowly than user tool. It does not make a sense to compare user tool to web / mid-tier based access. However, you may consider testing different browsers and browser versions.

Large part of client related problems and general performance loss may be related to the client's anti virus/ anti spam solutions. Perform analysis to determine if such conditions exist in your environment.

5.3.3. Patterns by Time

If you establish that certain problems occur in specific times create a weekly calendar and mark the specific times when the problems occur. Discuss with your team this calendar and find if your calendar entries coincide with other events in your environment.

Things to look for:

- On the remedy application side - escalations, scheduled reports, imports, exports, reconciliations.
- On the OS level - compare your calendar entries to maintenance windows, backups, restores, service restarts, etc.
- On the OS level - find out if the servers are hosting other applications, and if they are, find out what their scheduled job lists looks like.
- Look for database level exports/imports.
- Look for maintenance of integrated applications.

Generally, working with your team brings faster results that generating and analyzing logs or extensive monitoring where the tools are available.

Once you establish a list of correlated events find out what are the options of changing them. Some times it may not be possible.

Note: In one case customer found that every Thursday a scheduled SAP script runs and generate reports on the AR System application server. During that time the performance was so bad that no one could get anything done. The customer did not have an option to separate the applications, so, they marked Thursday from 4:00 PM to the end of the business day as a dead time and decided to document any calls they may get on paper and update the system on the following day.

Even if you don't have the option to resolve the problem(s) immediately, create a problem record(s) and document your findings, it is important for future reference and planning.

A variation of time related problems are the memory leaks related problems. Memory leaks cause the server processes to occupy more and more memory without releasing it entirely. As a result, your server processes can grow up to 20-30 times the size of memory that they normally use. Typical for this problem is that restarting of application services (processes) busts the performance, and then it slowly starts getting worse. Monitoring the size of memory used by the application service processes will give you clear indication for the presence of this problem. Make sure that you monitor the processes long enough, because during the routine run the memory usage goes up and down all the time. To establish the pattern you have to monitor the processes for a period of at least 24 hours, taking readings every half hour or so. If there are no patches available, scheduling of nightly restart can be used as a temporary fix.

5.3.4. Patterns by operation

You should concentrate on two arrears: SQL/process/Web Service related losses and workflow related losses. Frequently, a problem may be caused by a combination of both. You should have quite extensive list of user complains about poor performance related to specific application functions. Each complain have to be analyzed separately.

Start with reproducing each complain, generate combined logs of Active Link, API, Database, Filter on the client/server side. If the issue you are investigating is a "SAVE" operation, e.g. operation when entry is created or modified, use all four logs combined. If the issue is an operation performed without "SAVE", like table refresh, menu selection, window opening, you can limit your logging to Active Link and Database (eventually API).

If you are investigating "SAVE" operation, determine, using the log times, the start and completion times of Active Links processing and Filter Processing, then break it down in phases. This will give you some initial idea when the time is spent. Find out if there are SQL statements, $PROCESS$ calls of Web Services calls that consume large amount of time. You can compare the start time of selected action to the start time of the next action, it will give you the time that the selected action took to run. At the end of this analysis you should be able to say if there is few specific actions that cause significant time loss or the time loss is relatively evenly spread between all actions.

In the case of **evenly spread execution times** you have to deal with ineffective workflow,

In the case of **one or more actions that consume the majority of time**, you have to break the further analysis into action by action analysis.

There are several root causes that you have to classify every significant time loss into:

5.3.4.1. Ineffective workflow

Time consuming operation where the time is relatively evenly distributed between each action. It is a good idea in this case to keep the Active Link processing analysis separate from Filter processing analysis. Active Links are processed on the client side you may discover a client pattern there.

Ineffective workflow is in general very difficult to optimize, especially if it is out-of-the box several hundreds or thousands active link and filters application. Additional consideration is that modifying it may complicate your vendor support. Work with your vendor on documenting the problems and finding resolution.

Here are few possibilities to consider:

Check for endless loops when guides are called recursively.

Do not keep your disabled objects on the production server, it will slow down your overall processing speed.

Remove Fields and Workflow that your organization is not using. This action takes careful planning, investigation, and surgical precision. You may consider accomplishing such a project in several phases: disable the workflow, remove the fields, remove the workflow. It is strongly recommended that you use workflow analyzing tool to ensure that no other part of your applications uses the objects that you are going to remove.

There are several actions you can take toward optimizing the workflow.

Change the execution order of all objects that generate error messages to run first. This way you will avoid receiving error messages when most of the workflow is already executed. When you correct the error and repeat the action, everything will run from the beginning again.

Group active links and filters into guides. For example, if you have twenty filters running on modify when an entry is re-assigned ('Assignee' != 'DB.Assignee') you may consider creating a guide that runs all these filters in the same sequence, take out the "'Assignee' != 'DB.Assignee' from the filter conditions, and add a filter that checks if 'Assignee' != 'DB.Assignee', and executes the guide. You may say that now we have twenty-one filters instead of twenty, correct, however from execution point of view you have saved a lot. When the entry is not re-assigned only one filter will check if 'Assignee' != 'DB.Assignee' (instead of 20) and then will fail. When the entry is re-assigned only one filter will check if 'Assignee' != 'DB.Assignee' (instead of 20) and then it will execute the twenty filters in the guide. You gain performance in two places:

first, 'Assignee' != 'DB.Assignee' performs a database call, so you are reducing these calls from 20 to 1; second you are reducing the filter processing time because the filter conditions will be shorter.

In general, try to reduce the calls performing TR vs. DB comparison to one, either like in the example above or by using temporary field as a flag. Following the same example, add a temporary (Display Only) field character type with length one and default value "F" called Re-Assign. Add Filter (it can be implemented for active links as well, it is just quite different) checking if 'Assignee' != 'DB.Assignee', and if true ,setting Re-Assign = "T". Further in your workflow you can simply refer to 'Re-Assign'="T" instead of checking 'Assignee' != 'DB.Assignee' every time.

Avoid using calculation in queries and conditions. For example 'Resolved Time' > ($TIMESTAMP$ - 7*24*60*60) is a query that will find all the entries that have been resolved within the last 7 days. This will evaluate the expression ($TIMESTAMP$ - 7*24*60*60) for every comparison performed. You can save some resources if you calculate the expression before running the query. For example, create integer field TempInt and set it to ($TIMESTAMP$ - 7*24*60*60), then you change your query to 'Resolved Time' > $TempInt$.

These are actions that change your application significantly and gain relatively little in performance. Use them as last resort. It is always easier to take them in consideration when you are building new application then improving existing one.

There is a combination of both ineffective workflow and time consuming queries that I have noticed frequently and would like to bring to your attention. This is the case when you have forms containing multiple tables that take too long to display (load).

First, check in your AR User tool Tools->Options Advanced page if the value of Table Fields->Refresh Content On Display is checked (checked by default). If it is checked, un-check it. This option causes most tables to be refreshed twice when window is open. You can reset this value for all users if you are using centralized preferences, or create active link to reset it somewhere in the very beginning of the user session (opening a default home page is a good place to do so).

Most of the tables are refreshed by workflow. Check your active links to ensure that the refresh will run only once, and not on all Window Open, Display, Window Loaded.

Now, going one step further, most of the tables are not visible when you open the entry because they are on different pages. If this is the case you may consider changing the refresh action to take place when you click on particular page where the table is contained, rather than refreshing it in the beginning when the entry is displayed. This way, if the page is not selected, the table will never refresh and you will save valuable resources.

Last, but not the least important, chunk your tables. Take a look at the big search engines like Google and Yahoo, your search may find millions of entries, but they will display them for you in a series of 50's or so. If they display all entries found for each search in one bulk list their performance will be crashing as well. Generally, do not load more than 200 entries at a time. AR System has all the features to automate chunks with next and previous buttons, as well as to define custom queries that will reduce the search results to something more meaningful.

5.3.4.2. Single Action Delays

In all cases of **single action delays** you will find one or more of the following:

Single action time consuming **$PROCESS$** set fields. Log the process execution. Logon as the same OS user that executes the $PROCESS$ statement on the local machine or the server (where the process is executed by the application). Try executing the process manually from the shell/command line and if there are debugging tools available use them to find what causes the delay. Work with the provider of this executable and your OS admin to find ways to accelerate the process.

Single action time consuming **Web Services** set fields. Use whatever utility you have available to take a closer look at the web service launching and bringing the results back. TCPTrace.exe from PocketSOAP is very simple and effective tool for this purpose. Check with the Network Management for possible connectivity or DNS issues.

Single action time consuming **SQL** statements. You are entering very delicate area. Until now you were tracking times. You found numerous SQL statements that cause delays in time. Now, however, you are entering the world of cost. Your dba will most likely work with you in terms of cost, because all the decisions about how to execute a query and how effective a query is are cost based.

As a Remedy administrator you should be able to access some database management tools for your database instance. If you can't, work together with your dba. Prepare a list of statements that cause delays (you should have most of them identified in Step 4), discard the UPDATE, DECLARE, INSERT statements (if any) and concentrate on SELECT statements only. Ask your dba to prepare Execution Plan for each statement. Your first task is do determine if the statement performs full table scan or utilizes index. Always keep the cost associated with a statement and the count (frequency of run) together so you can effectively rank them.

What produces a SQL queries in AR System? Nearly everything, here is a list of the most important.

- The workflow engine constantly queries the AR System meta tables, you will notice statements to tables like actlink, arschema, filter, field, etc. It is not likely that any of these causes performance loss and you can just ignore them.
- Every time someone performs search or runs a report.
- Table refresh.
- Set field action from a form.
- SQL and Search menus.
- Push actions.
- …..

AR System will translate the selection criteria into SQL statement. You should be familiar with the structure of the underlying database. If you are not, take your time and read the "Database Reference" guide for your particular version and database.

Your first task is to associate each statement to specific action in your application. While the statements you found analyzing logs are easy to relate to particular active link (or filter), and based on that with an application action, the ones you find in the Top SQL list are pure SQL statements without any information who and how runs them. For example:

```
SELECT C1, C2, C3, C4, C5, C6, C7, C8, 0, C900, C902, C903, C904, C905,
C906, C907, C908, C909, C910, C911, C912, C913, C914, C915, C916, C917,
C918, C919, C920, C921, C922, C923, C924, C925, C926, C927, C928, C929,
C930, C931, C932, C933, C934, C935, C936, C937, C938, C939, C940, C941,
C942, C943, C944, C945, C946, C947, C948, C949, C950, C951, C952, C953,
C954, C955, C957, C958, C959, C960, C961, C962, C963, C964, C965, C966,
C967, C968, C969, C970, C971, C972, C973, C974, C975, C976 FROM T18
```

The most direct way to identify the application action is to find this statement in a log you have generated earlier. This will point you to the object executing the query, and therefore to application action.

If this is not an option, then find out what that statement means in AR System terms.

Let's start with the table. The 18 in T18 is the schemaid (in the past Remedy get used to call the form schema). You can query the arschema table to find the form name

```
select SchemaID, Name from arschema where SchemaID=18;
```

this will return:

```
18   Server Statistics (or some other form name)
```

Now you know that this query runs against the Server Statistics form.

You can then "translate" the C columns into field names and get clear idea what kind of information is retrieved, using statement like:

```
select FieldID, FieldName from Field where SchemaID=18;
```

This will return the list of all field ID's and field names from the Server Statistics form. Using this list you will find that C1 represents the field Request ID with field ID 1, C7 is Status, etc. Or you can open the Server Statistics form in Admin Tool and look in the field properties for each field ID.

Either way, at this point you will have clear idea what does this query do from AR System perspective. You can use an workflow analyzing tool to map this statement to an object and application action.

If the query performs a **full table scan**, take a look at the Form Properties->Indexes to find if the corresponding AR System form has appropriate indexing. If this is high cost frequently run query discuss with your dba what would be the best way to accelerate it.

If the form has indexes created, but none is applicable to this query discuss with your dba ways to accelerate it.

If the query does **utilize an index** and it is still too costly, again, discuss with your dba ways to accelerate it.

Since each query optimizer is different, there is no common rule how to index your table for best performance. Work with your dba to explore the available options. Keep in mind that all indexes has to be created or modified using AR Admin tool to ensure the integrity and portability of your applications. You can always add and remove index, just bare in mind that maintaining indexes has it's cost as well. In this relation, do not index small tables. (if the optimizer estimates that the table will be read as one block, it will most likely decide to perform full scan). Use Execution plans to prove your concepts before taking action.

DO NOT BE AFRAID TO MAKE CHANGES IN YOUR INDEXES. Just do it in orderly manner, prove your concept, test, be prepared to roll back, work with your dba or someone who has better understanding on how indexes work. Do not take anything as given. In some occasions only differences in data could cost big differences in behavior of otherwise absolutely the same applications. With simple words, what works for Joe may not work well for you, even if you and Joe have absolutely the same products. That's why you shell perform your own application fine tuning.

Through the years I came to few very common weaknesses that I have seen in both Oracle and MS SQL implementations. Be aware that the query optimizers are quite different for each database as well as the different releases of the same database. Test before making your mind. There are lots of articles and references available on the net, so if you want to find out how your particular optimizer makes decisions, just run a search and do some reading.

Things that you should be aware of:

The order of the parameters in your query or condition does not affect how the query will be executed. For example:

'Status' = "Resolved" and 'Assigned to Group' = "Remedy Support" and 'Closed Time' > ($TIMESTAMP$ - 7*23*60*60)

will be executed in the same way if you changed to:

'Closed Time' > ($TIMESTAMP$ - 7*23*60*60) and 'Status' = "Resolved" and 'Assigned to Group' = "Remedy Support"

If you run the same query several times in a sequence you may see different completion times. This may be result of loading some or all of the results in the cache (especially in poorly populated test environment).

After recreating an index a query may run significantly faster, then in few days or so, becomes slow. In Oracle (and I believe in some MS SQL versions as well), if you drop an index the database does not create the table statistics immediately. Until the statistics become available the query optimizer will operate in rule mode, where come queries, like C7<4, could run faster. Once the statistics are gathered, the optimizer will switch in cost mode again.

Only one index is used when a query is executed.

Things you should check for right away:

C7<4, which generally represents 'Status'<"Resolved" if the Status has 6 values and the last 2 are Resolved and Closed, will most likely perform full table scan regardless if you have created index on Status or not.

Work around: create an index on Closed Time, then change the query to Closed Time=null. It will return all open items. All this is possible if your application sets Closed Time = some date/time when the Status is changed to Resolved. If you decide to use it you will have to add a filter to reset the Closed Time to null (if the application does not do that) when an entry is re-opened (for example, when Status is changed from Resolved to Assigned).

"%"+ 'Assigned to Group' + "%" LIKE $GROUPS$. LIKE always results in full table scan. In the case of $GROUPS$ it is even worse. $GROUPS$ contain all the groups an user is member to. This includes permission groups as well. There is no way that an entry can be assigned to a permission group, so the comparison process can be significantly shorter if we do not compare against the permission groups.

Work Around: If you are not using calculated groups the following is a really good approach. Using workflow create the query as following: 'Assigned to Group' = "Remedy Support Team" OR 'Assigned to Group' = "Oracle Support", where the "Remedy Support Team" and "Oracle Support" are all support groups that the current user belongs to. Use EXTERNAL() to run the query.

Indexes like Create Time + Status does not work well because Create Time more or less is unique, better use Create Time only.

If you establish that some indexes are not used at all, just drop them, you will save the time for maintaining them.

Keep in mind that execution time is something very relative to be used as measurement, when you are analyzing and comparing sql queries always use cost.

5.3.4.3. Large Amounts of Data

The quantity of data stored in the application forms affects the performance significantly.

There are two courses of action to reduce these amounts:

– Archiving and of-loading some or all of the reporting on the archived data set.

– Pruning – cleaning the data once it becomes obsolete.

Both actions are discussed in separate article "Reporting Performance with AR System Applications.

6. STEP 6: IMPLEMENTING CHANGES

By now you should have quite extensive list of problems with completed root cause analysis. If it is possible, rank them by severity.

Attach the possible solutions to each problem. Add an estimate (in terms of human, financial and other resources) what will take to implement each solution. Add an indicator if this estimate requires internal resources, external resources, or both.

Now is the time to go back to administrative action. Meet with all affected and involved parties and find out what solutions can be implemented immediately, what solutions will be put on hold (and for how long), and what solutions will not be implemented at all or will be implemented in a very far future.

For the latter, create problem records, document all findings, document the possible solutions, as well as the considerations for not implementing them. Please, note, that these are real problems that will exist in your environment for long period of time.

For the ones that will be implemented immediately or in near future, create a project plan(s) and make sure that all resources are approved and will be available on time.

Execute the project plan(s), do not stop in the middle of implementation if you have achieved significant improvement in performance. Fix as many problems as possible, because, with the time passing your environment will be changing and your current findings may not be valid in the future.

7. TIMEFRAME

The whole process Step 1 to 5 usually takes 2-6 weeks with one person dedicated to the project 100% and team members utilizing 25-50% of their time on it. This is really very relative estimate. It depends on the environment, available tools, nature of problems, dedication of team members, ability of management to make quick decisions.

After this initial period you should review the possibilities. make decisions what will be implemented and when, set timeframes, plan resources. This should not take longer than 2 weeks.

The actual implementation will take whatever you have determined in your project plan(s) from Step 6.