Proactive Performance Management Using Analytics

Steen Rasmussen, CA technologies
Principal Engineering Services Architect
Abstract

- **Analytics as a Weapon to Proactively Manage DB2 Performance**
- Are you struggling to manage Db2 performance because you don’t have established baselines that illustrate normal behavior? Do you spend too much time in reactive mode? What if you could pinpoint “a creeping trend” before an SLA is violated? Discover how you can quickly identify a sustained deviation from the norm by predicting the performance of your static SQL programs and critical dynamic SQL transactions. See how you can avoid the “war room” and put yourself in a proactive position to take corrective action(s) before problems occur.
First a brief overview of how analytics were viewed and used a few years ago, then what we have seen in the market the past few years like IBM Watson being one example and finally where I see analytics going within the next year in terms of being used in the IT environments.

There are so many terms being used within IT analytics – but also in general terms, so we will cover some of the most common terms.

You can either buy analytics solutions for your IT needs or you can build your own based on how sophisticated you want to be and the skills available. We will cover a couple of different methods to consider.

Throughout the presentation we will discuss what baselines is and how to use baselines as well as the pro’s and con’s.
Analytics Yesterday, Today and Tomorrow
Analytics Yesterday

- Nothing new – except maybe for z/OS performance
  - Buying patterns resulted in reorg of grocery stores
  - Data mining using DW technology
  - Offloading data and analyze performance metrics

- What has changed
  - Can’t wait hours, days or weeks
  - Need to see deviations near real time
  - Will require streaming of performance metrics

- Why is it changing
  - Apps are not what they used to be
  - Demographics, lack of SMEs, skills etc.
  - Need to recognize a problem automatically and alert => react quickly to problems that matter => ultimately guide to a solution

Just a few years ago, analytics wasn’t really associated with IT performance. Shops used transactional data, logs etc. to offload to data warehouses, data marts etc. – and typically to another platform than the mainframe for analysis and data mining – often used to find buying patterns in order to rearrange the grocery store for example.

Thinking about the APP ECONOMY, nobody can wait weeks or days to get the results from analytical processing – and especially when it comes to application performance data – basically we need to stream performance metrics in order to catch anomalies almost before they happen.

The applications are not updated on a Wednesday night anymore – new versions are implemented almost on an hourly basis in order to stay competitive. The days are over where the DBA’s knew all the applications and the “normal behavior” – the next MF generation has started and lack some of the knowledge. The need is here to automatically identify the normal behavior and catch anomalies and potentially provide a solution.
Analytics Yesterday

Is this a typical scenario?
Information Overload Leads To Poor Decision Making / No Decision

The command center with dozens of screens, applications etc is not uncommon. The issue is INFORMATION OVERLOAD where you can’t see the forest for trees which can lead to poor or no decision being made.

We see red or yellow alerts all over the place due to manual and static defined thresholds, so we just say “this is always red at 10am”

There’s indeed a need to filter the noise and focus on the real problems – trying to get out of the reactive mode.
Which issues / challenges to be addressed

- **Why Analytics**
  - Limit “war room scenarios” and finger pointing
  - Reduce MTTR and identify root cause quickly
  - Ultimately predict the future to mitigate problems
  - Do you know WHAT the normal behavior is
    - System wide STATIC thresholds from monitors are not sufficient
    - Have you established baselines in order to take action before a problem gets out of control
    - For Db2 packages, dynamic statements ?
    - How are you maintaining the baselines when “the world changes”
    - Are you using STATIC baselines and getting loads of False Positives ?
  - Is the “green highway” defined based on all the metrics ?
  - What about calendaring events ?
  - Typical Monitors identify and alert one-time events

The challenges every IT environment are looking at solving are numerous:
Eliminate the need for the war room meetings when performance issues arise and reduce the Mean Time To Resolution.
Once enough statistical information is available, being able to predict what will happen when certain metrics are correlated.
First of all – you need to know what the “normal behavior” is and static performance thresholds are not the answer – you need to baseline static as well as dynamic SQL, you need to consider how to generate the baselines, calendaring events, identify how the application behaves based on weekday, hour etc.
Analytics Today . . . and Tomorrow

- Prevent “False Positives” and only alert on problems which matter – using the Green Highway and machine learning doing continuous streaming of performance metrics ultimately alerting before an event happens.

One of the most important issues is to avoid false-positives and only alert when there is a real problem. Introducing “the green highway” to understand how an application behaves throughout the day and week – what is the most likely, less likely and unlikely behavior is a must in order to not generate false-positives and volatility is a key factor. Looking at the lower left hand graph, what the normal behavior is between 10am and noon is different then the behavior at 10pm – making it very important to have some level of calendaring event mechanism embedded.

For automation purposes, being able to alert is important – but not only alert, also provide the receiver of the alert with sufficient information in order to make quick decisions and take corrective actions immediately.
Analytics terms, algorithms, “buzz words” and explanations
Western Electric Rules (from Wikipedia)

- In Statistical Process Control, the Western Electric Rules are decision rules for detecting "out-of-control" or non-random conditions on control charts. Locations of the observations relative to the control chart control limits (typically at ±3 standard deviations) and centerline indicate whether the process in question should be investigated for assignable causes. The Western Electric Rules were codified by a specially-appointed committee of the manufacturing division of the Western Electric Company and appeared in the first edition of its Statistical Quality Control Handbook in 1956. Their purpose was to ensure that line workers and engineers interpret control charts in a uniform way.

Analytical processing often mentions Western Electric Rules. At first view you might wonder what this has to do with performance metrics and anomaly detection, but reading the Wikipedia definition (especially the STANDARD DEVIATION part) will start to open the eyes, so let’s have a closer look at what the standard deviation is – especially related to BASELINES and the “normal behavior”.
Western Electric Rules

This illustrates a static baseline pretty well: the solid X-line. The standard deviation measures how far away a data-point is from “the norm” and the top graph illustrates an anomaly (red dot) since it’s more than three standard deviations away from “the normal behavior”.

Looking at this with Db2 performance glasses, this kind of event could be captured and handled by a performance monitor based on a static threshold. In order not to get false-positives, we need to look at SUSTAINED DEVIATION – meaning multiple consecutive data-points being more than three standard deviations away.

Another issue to consider related to Db2 performance metrics – what if a data-point falls below (meaning performance is improving) – do we need to be alerted of these as well?? Probably since it could be due to new hardware, Db2 release etc. so a need to RE-BASELINE could be a good idea.
Moving Average

- Unlike STATIC baseline, the actual metrics are “smoothed” for DYNAMIC baselines

Moving averages smooth the price data to form a trend following indicator. They do not predict price direction, but rather define the current direction with a lag. Moving averages lag because they are based on past prices. Despite this lag, moving averages help smooth price action and filter out the noise. They also form the building blocks for many other technical indicators and overlays, such as Bollinger Bands, MACD and the McClellan Oscillator. The two most popular types of moving averages are the Simple Moving Average (SMA) and the Exponential Moving Average (EMA). These moving averages can be used to identify the direction of the trend or define potential support and resistance levels.

Here’s a chart with both an SMA and an EMA on it:

If a STATIC baseline was used – and April 4-11 stock price was used to calculate baseline, an alert would have happened April 15 while the moving average would probably trigger it a little later.

You can have STATIC baselines which basically is the average metric over a period of time. This can lead to false positives depending on how you handle incoming new metrics – and especially how many consecutive metric violations you decide to happen before alerting.

One method to limit false positives is using the MOVING AVERAGE which will smooth “bumps” and change the baseline dynamically. Basically, if you have small increments over time, the moving average will (in many cases) not trigger an alert - unless the increments are relative high. The downside is you might discover an anomaly later or potentially never – again it’s a trade off based on your goals.
Sustained Deviation

• You all have monitors keeping an eye on the health
• Most of you have static thresholds defined
  • Some don’t due to “false positives”
• Good baselines will help
  • Don’t report on “one time-off events”
  • Report on a number of consecutive off-the-norm incidents
    • One example: a DB2 package or SQL statement is using 15ms CPU instead of 10ms CPU
    • Do you really want to be notified unless it’s keep happening?
• Look for Sustained Deviation:
  • Let’s assume you stream performance metrics every xx minutes
  • How many “intervals” do you want to violate baseline before alerting?
  • If hourly intervals maybe 2 intervals – if 10 minute intervals maybe 3 or 5?
• Important to look into both statics and dynamic baselines

We just talked about baseline metrics being violated and how many consecutive data points you want to consider before alerting. This is also known as sustained deviation – since we don’t want to alerts because of ONE SPIKE – that’s why we have real time performance monitors.

Basically you don’t want to mimic what the performance monitor is doing since you will drown in false-positives. How many consecutive data points going off the green highway/baseline is something you have to consider.

Based on the moving average smoothing metrics over time, and if you really do want to catch the “creeping trend” you might want to correlate both the static and dynamic baseline is one way or the other.
Baseline Consideration

- What about eliminating “outliers”
  - Will narrow the “green highway” and could cause “FALSE POSITIVES”
    - Can be mitigated by requiring a number of consecutive events moving outside before creating an alert
  - Remember you have monitoring tools catching “one stand off’s”
  - We are looking for sustained deviation
  - What is the goal?

Once you start to create your baseline, there are a few things worthwhile to consider. Do you want to eliminate the OUTLIERS from the baseline calculation? The issue is, if you don’t remove any of the outliers, your green highway may be so wide that you don’t catch any anomalies. On the other hand – if you remove too many you start getting false-positives, so it might be an art to find the right balance.

This is where the more sophisticated algorithm can come to the rescue since you can divide the performance metric into MOST LIKELY, LESS LIKELY and UNLIKELY – and potentially correlate this with time of day, day of week etc.
Baseline Consideration

- What about eliminating “outliers”
  - Will narrow the “green highway” and could cause “FALSE POSITIVES”
    - Can be mitigated by requiring a number of consecutive events moving outside before creating an alert
  - Remember you have monitoring tools catching “one stand off’s”
  - We are looking for sustained deviation
  - What is the goal?

On the other hand, you might get too many false-positives when sustained outliers appear in the data.

When you compare the previous picture with this one, you can tell that removing more outliers will create a more “aggressive” baseline and potentially drown you in anomalies.
Static or Dynamic Baselines

- This is where some hard decisions are needed – what is the goal
  - This transaction has small increments of CPU increase over time (more details later)
  - Using a STATIC baseline will quickly capture this anomaly before it goes too far beyond the baseline
  - A DYNAMIC baseline using 7-day moving average will smooth the increments and might not catch this anomaly – unless more sophisticated algorithms used

Talking about static/dynamic baselining and also the moving average – which one is the best ??
Let’s have a look at this real-life example:

1) The white space to the left is the period of time metrics were collected and used for the baseline.
2) The baseline (static) is the dotted line in the blue bar – and the blue bar illustrated the standard deviation used for this baseline.
3) The zig-zag graph illustrates the actual metrics streamed.
4) The solid darker graph illustrated the 7-day-moving-average

As you can tell, the CPU-TIME has small increments over time and the 7-day-moving-average smoothes this over time for the dynamic baseline and never issues an alert while the STATIC baseline identifies a sustained deviation and alerts before the problem becomes too critical.
How Do You Create Algorithms

• Depends on how sophisticated you want to be.
  • There are plenty to look at from the internet.
    • Even XLSX can help out.
  • Machine Learning is a different “beast”

The metric for 7-day moving average could be calculated the following way:

\[
\text{SELECT} \left( \frac{\text{SELECT SUM(numberOfX) FOR PREVIOUS(dateOfX, 1))} + \text{SELECT SUM(numberOfX) FOR PREVIOUS(dateOfX, 2))} + \text{SELECT SUM(numberOfX) FOR PREVIOUS(dateOfX, 3))} + \text{SELECT SUM(numberOfX) FOR PREVIOUS(dateOfX, 4))} + \text{SELECT SUM(numberOfX) FOR PREVIOUS(dateOfX, 5))} + \text{SELECT SUM(numberOfX) FOR PREVIOUS(dateOfX, 6))} + \text{SELECT SUM(numberOfX)} } } \right) / 7
\]

There are plenty of algorithms available on the internet, MS EXCEL has several calculations available and you can probably find a lot of open-source capabilities as well.

If you want to take the extra step and use ML / AI, that’s a different story.
How to get there - Where to start
You Need Performance Metrics

- Several sources to choose from
  - SMF data
  - IFCID trace records (for example 316, 401)
  - SQL Monitors
    - CA Detector
    - BMC Apptune
    - IBM Query Monitor
    - . . . . and there are others

- Which performance metrics?
  - Some of the obvious ones:
    - INDB2-CPU and IN-DB2 ELAPSED
    - #GETP, SYNC/ASYNC I/O
    - Wait counters
    - #Executions

Without performance metrics we will get nowhere and there are many various sources to use,

Since this presentation is focused on SQL application performance, there are some obvious metrics to look at:

*) INDB2-CPU and ELAPSED since these metrics correlated with #executions can be used in various ways.

*) Number of GETPAGE’s as well as synchronous and asynchronous I/O can help us track differences in access path, disorganized objects, unclustered inserts etc.

*) Db2 provides a bunch of wait-counters, maybe you want to focus on a few of these.

*) some Db2 shops want to look at how frequent a plan/package/statement is executed since these can impact other metrics in the environment
What Do You Want To Monitor

▪ Static SQL
  • Metrics on the Plan and Package level
  • Why not on the statement-level?
    • Program changes, STMT# changes, added/removed statements will force re-baselining

▪ Dynamic SQL
  • Normalize / Concentrate to better compare “identical” statements?

▪ How frequent
  • Probably not per transaction/plan/package execution (think about DISTSERV)
  • Aggregate selected performance metrics on a frequent basis
    • Once a minute / once every 5 minutes / once every hour?
    • Depends on how quickly you want to detect anomalies

Next question is what type of SQL transactions you want to monitor?

For static SQL it makes sense to look at the Db2 performance metrics for plans and packages but not so much on the statement level for various reasons:

    The program changes
    SQL statements are added/removed/modified

So in order not to have to re-baseline all the time, I personally prefer to look at the behavior from the package level and perhaps from the plan-level depending on how the plan concept is used.

For dynamic statements – you might consider eliminating this type depending on what you want to accomplish. If you do want to monitor dynamic statements I recommend “concentrating” these in order not to drown in data.

Finally – how frequent do you want to stream data? Probably not on the transaction-level, but choosing a fixed period of time where the metrics are aggregated makes sense.
How Much Data is Needed

- Historical data needed in order to establish a baseline:
  - As much as possible – the more the better due to accuracy
  - What about calendaring events – and the level of sophistication
    - 24x7, 8am to 5pm, weekday, workload differences, weekends, Black Friday, ...
    - What / where is the “normal behavior”
  - Focus on top consumers / most important transactions?
  - In today’s app economy everything changes all the time
  - Don’t limit yourself – a transaction with few executions today might be the hitter tomorrow.
  - If you don’t baseline everything – nothing to compare to when “things gone South”
  - You might have to collect performance metrics for some time before baselining

Another tough question – how much data is needed when creating the baselines?

Again many issues to consider:

*) If you want to find the normal behavior for Black Fridays you need a lot of data.
*) Are you only interested in certain timeslots of the day and days of the week
*) Do you know when “the normal behavior” is?

Some Db2 shops prefer to only look at the top consumers but in today’s dynamic world everything changes constantly so baseline everything instead of having to re-baseline all the time when the transaction mix changes.
Other Issues To Consider

- Which transactions, batch jobs etc.
  - Only top 50 – but what is top 50?
    - Number of executions
    - Resources consumed
  . . . . . .
  - What about the fastest best performing transactions (think of ATM) – you really want to focus on these too (live example to follow)
  - You might be missing opportunities (performance) if you limit yourself
  - If you don’t have the historical metrics for everything
    - You will have to BASELINE all the time due to changes in workload
    - What is important today doesn’t exist tomorrow and visa versa
  - Does a transaction always behave the same?
    - When a holiday Monday is a payday
    - Change in behavior due to workload (BP-hitratio, locks, waits, . . .)

- Baseline EVERYTHING ! ?

If you for whatever reason only want to look at the top xx consumers, are these the most important ones, the ones consuming the most CPU, the most frequent executed ones, . . . . . .

Bottom line, don’t cut it short and don’t limit yourself – baseline everything so you even know the normal behavior for the transaction which barely executes today but will be a hitter two months out.
How To Do It – eliminate the manual processes

▪ The EASY way
  - Save the data for what you consider a “normal” timeframe
  - Use the AVERAGE calculations and create a STATIC baseline
  - Once baseline established - for every chosen “interval” (simplistically) :
    - Add the new data to the repository
    - Validate the metrics against the baseline
    - Add ONE to the Alert-counter for those plans/packages/statements
    - If Alert-counter is 3 – send an alert

▪ The HARDER way
  - Every time new data is added, the “Moving Average” needs to be calculated using appended data and historical data - and compared to where the new data “lands”

▪ The ULTIMATE way
  - Using ML / AI technology to do everything
  - Implement calendaring events etc. . . . . .

So how do you get there – once all the decision have been made covered on the previous slides?
Next thing to consider is the amount of effort you can afford.

You have to save the data from the period you would like to consider “normal” and use this as the LEARNING period. For the STATIC baseline, simply use the SQL AVERAGE function to calculate the baseline for the metrics you are interested in. Based on the frequency you want to check for anomalies, stream the data and use the new data points as an additional series of metrics – and compare to the baseline. If a metric/application falls outside the “normal behavior” add ONE to a counter and if you have eg. THREE consecutive intervals of data points violating the baseline then create an anomaly alert.

As you can tell, if you want to use the moving average, the calculations become a little bit more tricky.
Some Real-Life Scenarios
Correlating Metrics Can Complete The Story

CPU and GETP increase – correlating this information with RTS illustrated a lot of UNCLUSTERED INSERTS. WAIT and IO times have different behavior depending on the weekday.

Without historical information at hand, you could be focused on WAIT-times since it spikes occasionally, since the GETP and CPU increase is not alarming (YET)

Self explanatory
CPU suddenly went up but most other metrics decreased a bit (focus on 7-day moving average). It turned out a change was made to a SQL statement. When only CPU increases and no other metrics – a good indicator to look at the application / SQL.

Self explanatory
Correlating Metrics Can Complete The Story

Most metrics spikes the same day of the week - and same timeslot – looking at #executions gave the answer – maybe this is “just the normal behavior” due to transaction traffic.

Self explanatory
Correlating Metrics Can Complete The Story

A couple of different scenarios:

1) GETP creeped up and also CPU and wait-times
   • Further analysis indicated the GETP increase was for an index
   • Index Page Splitting was the reason
   • Online Reorg index scheduled weekly

2) A sustained deviation was identified during the weekend
   • Execution rate over the weekend was much lower than weekdays
   • Thankfully before the busy Monday morning
   • An alert was created that CPU constantly was outside the norm
   • Apparently a program was updated with modified predicates with insufficient RUNSTATS parameters (FREQVAL and COLGROUP)

Self explanatory
Baselining and considerations
What is the “Green Highway”

- “white space” is the learning time
- “dotted line” represents the STATIC baseline
- “blue is actual metrics streamed on a periodic basis
- “black” is moving average

For a static baseline, you can define the Green Highway based on how many standard deviations you’re interested in. For a dynamic baseline, the highway will obviously vary based upon the metric spread.

To recap some of the technical terms and the essence of this presentation:

1) Identify the learning period of time needed
2) Decide whether you want a static or dynamic baseline – or maybe both
3) Decide how frequent you want to compare the data points with the baseline
Static <> Dynamic Baselines

- Which one is best? “IT DEPENDS”

- Static Baselining (Dynamic Thresholds)
  - Analyze time series of data + calculate average => “normal behavior”
  - **PROs**: small increments of metrics can be caught ahead of time before SLA is violated (aka the Creeping Trend)
  - **CONs**: Small variations can cause “false positives”
    - Need to Reevaluate baseline when the world changes

- Dynamic Baselining
  - Continuously learning based on historical data
  - Adjusts “baselines” automatically (moving average)
  - Greatly reduces “false positives” and handles time dependent variances

Looking at all the scenarios throughout this presentation, you have some hard choices / decisions to make regarding what type of baseline fits your environment the best and the amount of labor you can afford to put into a project like this.

Another important issue to think about is when the metrics are constantly higher/lower than the baseline – is this due to an application change, CPU upgrade, z/OS upgrade, new HW - maybe it’s time to re-baseline.
Thank You - Questions?
Steen Rasmussen
Principal Engineering Services Architect
Steen.rasmussen@ca.com

www.ca.com