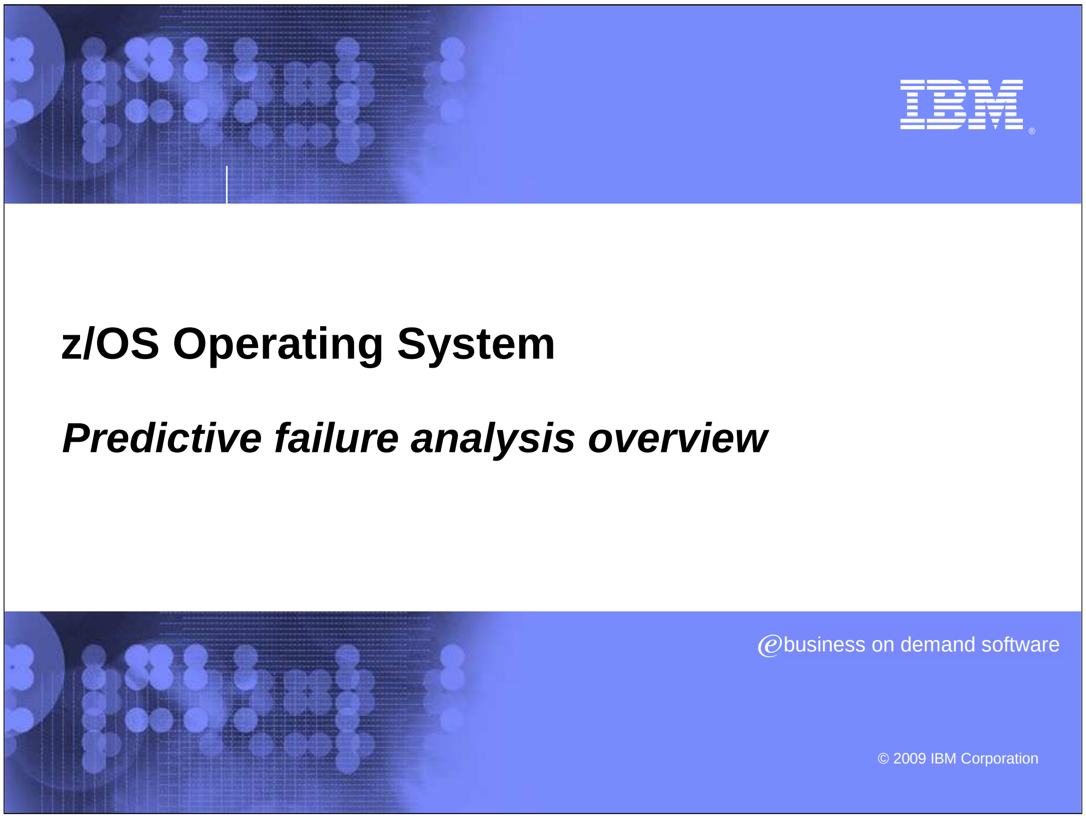
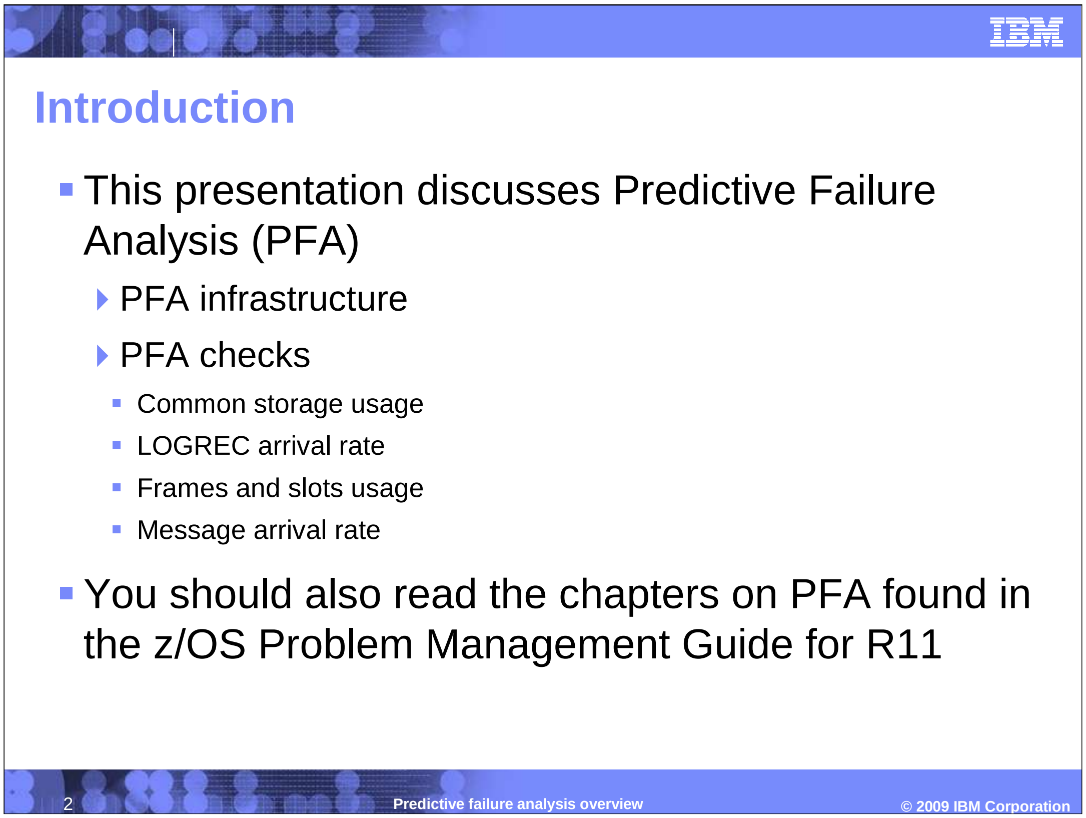


z/OS Operating System

Predictive failure analysis overview

@business on demand software

Predictive Failure analysis is a new component on z/OS® which was shipped on z/OS V1R10 as an SPE. It provides a way to detect soft failures, otherwise know as “sick, but not dead” incidents. It uses IBM Health Checker for z/OS remote check support to provide this function.



Introduction

- This presentation discusses Predictive Failure Analysis (PFA)
  - PFA infrastructure
  - PFA checks
    - Common storage usage
    - LOGREC arrival rate
    - Frames and slots usage
    - Message arrival rate
- You should also read the chapters on PFA found in the z/OS Problem Management Guide for R11

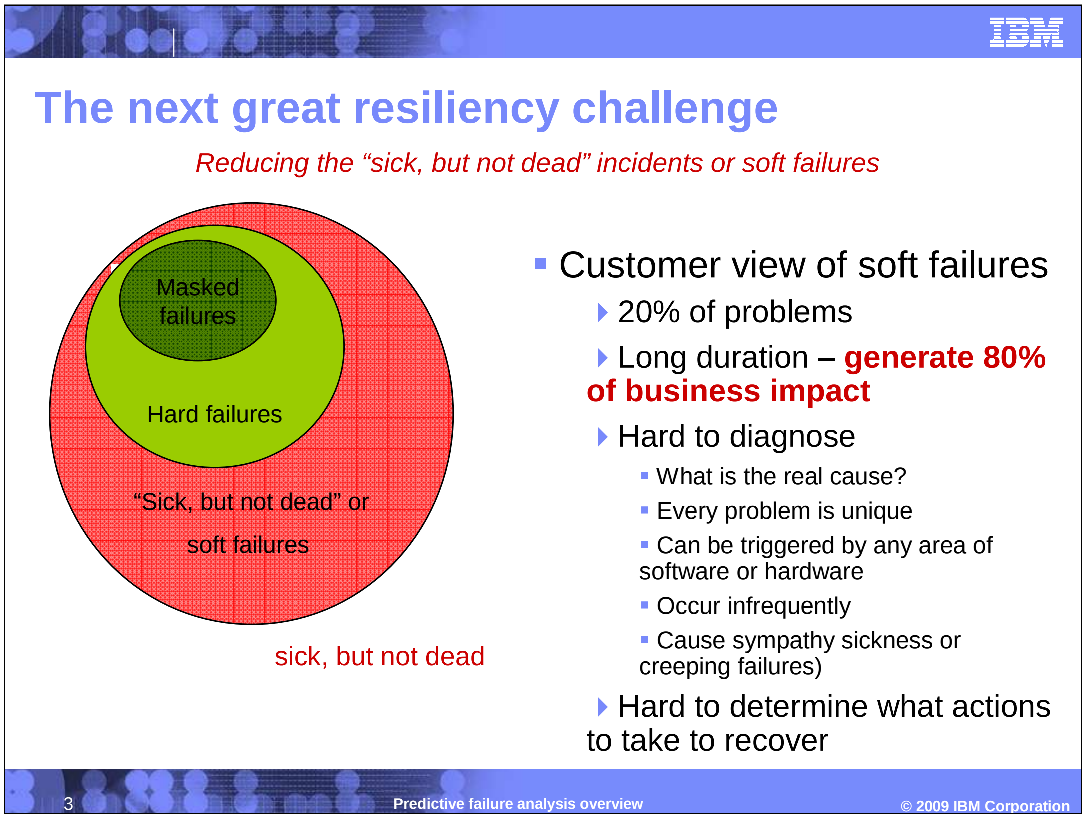
2 Predictive failure analysis overview © 2009 IBM Corporation

This module will provide an introduction to PFA. It will discuss why it was needed and what it is intended to do.

It will also cover the four checks that are available with PFA and the differences between PFA's checks and other remote health checks.

Predictive Failure Analysis is a new component of z/OS that was made available as an SPE in March 2009 for R10. The first two checks, Common Storage Usage and LOGREC Arrival Rate, were made available for R10 and the second two checks, Frames and Slots Usage and Message Arrival Rate are available starting with R11.

The highlights of PFA and the checks available are discussed in this module. Since this presentation does not cover all of the details of PFA, you should also read the PFA chapters of the z/OS Problem Management Guide which was updated with PFA information for R10 and was made available on-line at the end of March, 2009. That version of the PFA documentation contains information on everything that is available with the R10 SPE. PFA enhancements for R11 are available in the R11 version of that document.


IBM
Masked failures
Hard failures
"Sick, but not dead" or
soft failures
sick, but not dead

The next great resiliency challenge

Reducing the "sick, but not dead" incidents or soft failures

- Customer view of soft failures
  - 20% of problems
  - Long duration – **generate 80% of business impact**
  - Hard to diagnose
    - What is the real cause?
    - Every problem is unique
    - Can be triggered by any area of software or hardware
    - Occur infrequently
    - Cause sympathy sickness or creeping failures)
  - Hard to determine what actions to take to recover

3 Predictive failure analysis overview © 2009 IBM Corporation

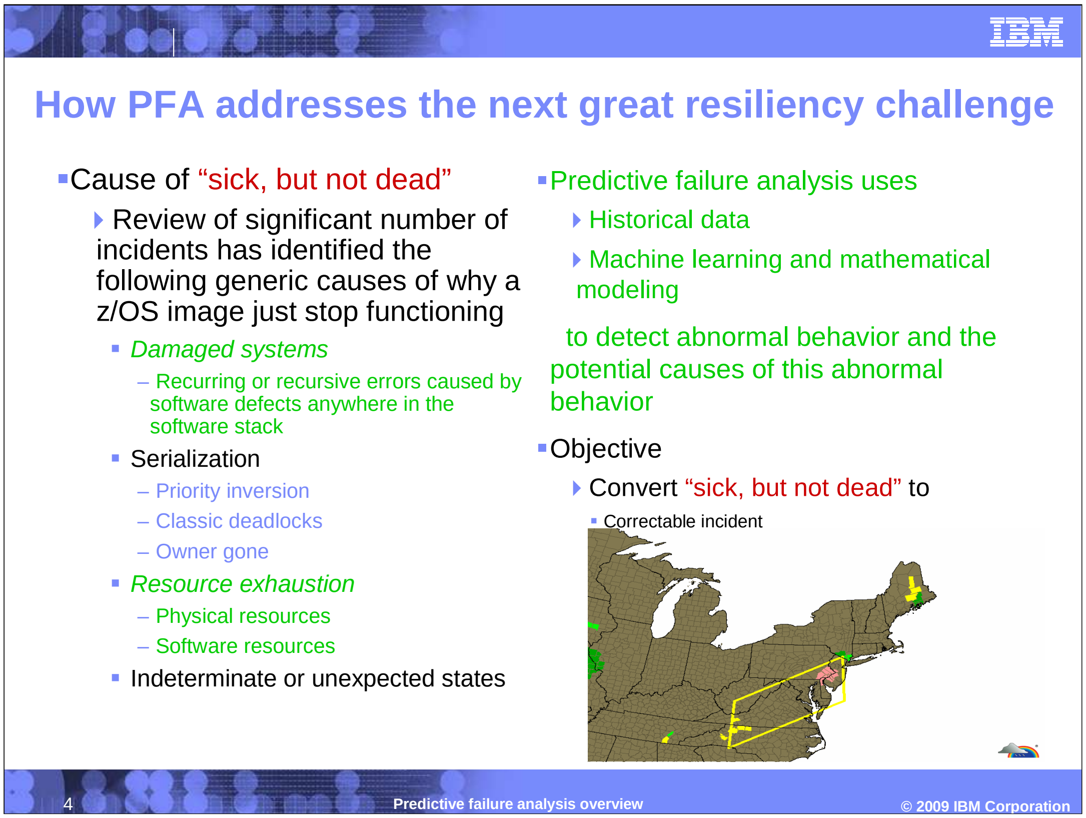
There are three general categories of software detected system failures: masked failure, hard failure, and failure caused by abnormal behavior. A masked failure is a software detected system failure which is detected by the software and corrected by the software. A hard failure is when the software fails completely, quickly and cleanly. For example, a hard failure occurs when an operating system kills a process.

A system failure caused by abnormal behavior is defined as unexpected, unusual, or abnormal behavior which causes the software solution to not provide the service requested. This abnormal behavior of the software combined with events that usually do not generate failures produce secondary effects that may eventually result in a system failure. These types of failures are know as soft failures.

These soft failures are a small percentage of the problems when compared to masked failures and hard failures, but they cause most of the business impact.

They are hard to diagnose due to the fact that the failure likely does not occur in the address space causing the problem, but more likely occurs in another address space. This sympathy sickness has been observed when either hard failures or abnormal behavior generates a system failure which could not be isolated to a failing component or subcomponent. Failures caused by abnormal behavior often generate sympathy sickness where the problem escalates from a minor problem to the point that the service eventually stops working. Because they are difficult to detect, are very unique, can be triggered anywhere in either software or hardware, and occur infrequently, failure isolation is very difficult.

Hard failures are deterministic in nature. However, a failure caused by soft failures is difficult to recognize within the component and are probabilistic and depend on secondary effects to cause observable damage.



How PFA addresses the next great resiliency challenge

- Cause of “sick, but not dead”
  - Review of significant number of incidents has identified the following generic causes of why a z/OS image just stop functioning
    - *Damaged systems*
      - Recurring or recursive errors caused by software defects anywhere in the software stack
    - Serialization
      - Priority inversion
      - Classic deadlocks
      - Owner gone
    - *Resource exhaustion*
      - Physical resources
      - Software resources
    - Indeterminate or unexpected states

- Predictive failure analysis uses
  - Historical data
  - Machine learning and mathematical modeling

  to detect abnormal behavior and the potential causes of this abnormal behavior
- Objective
  - Convert “sick, but not dead” to
    - Correctable incident

4 Predictive failure analysis overview © 2009 IBM Corporation

Our analytical understanding of this problem is by inference, not by direct measurement. Based on analysis and the input from subject matter experts, this area accounts for between 15-30% of problems that impact business.

A detailed study of all of the customer-reported problems from a set of our largest customers was performed. That analysis found that single system outages and single application outages were more prevalent than multiple system outages and multiple application outages. As part of this analysis, we investigated how long it took from the customer was first aware of a soft failures until an outage actually occurred. Some, but not all, of the these outages which occurred on a human time scale were observed when the system just seemed to stop functioning.

In these scenarios, the system stayed up, but was not processing work. For example, the system was unresponsive because of recurring failures in a CICS® transactions or the system was unresponsive because of a hardware failure causing IOS to go through recovery to re-establish a path to the DASD and shortly after re-establishing the path, it failed again.

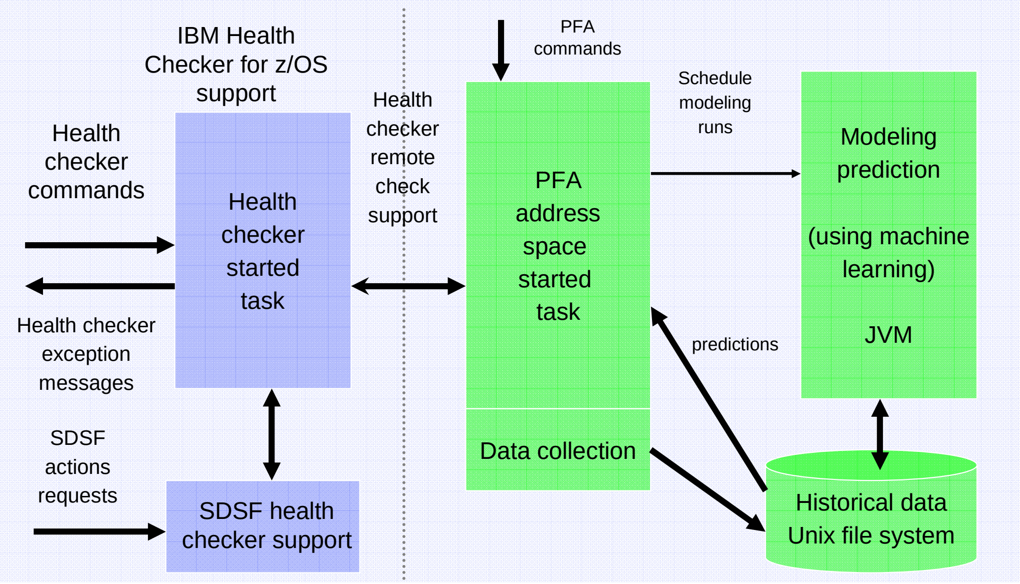
In general we think these errors fall into categories like recovery, logical errors, over-consumption, and sympathy sickness.

After reviewing many incidents, it was determined that there are four generic causes of why the system stopped functioning. The first is a damaged system. The indication of a damaged system is typically when there are recurring or recursive errors anywhere in the software stack. The second type is serialization. Serialization problems are most often caused by priority inversion, classic deadlocks, and owner gone scenarios. The third type is physical or software resource exhaustion. The last type is indeterminate or unexpected states.

PFA addresses this resiliency challenge by focusing on the damaged systems and resource exhaustion categories. PFA uses historical data along with machine learning and mathematical modeling to detect abnormal behavior and the potential causes of this abnormal behavior. PFA’s objective is to convert soft failures to correctable incidents.

PFA is somewhat analogous to the National Weather Service. Based on historical data, we predict when a soft failure might be occurring, alert the system operator of the situation, and provide information to assist in resolving the problem.

PFA infrastructure



IBM Health Checker for z/OS support

Health checker commands

Health checker started task

Health checker exception messages

SDSF actions requests

SDSF health checker support

Health checker remote check support

PFA commands

PFA address space started task

Data collection

Schedule modeling runs

Modeling prediction

(using machine learning)

JVM

predictions

Historical data Unix file system

It is important to understand the relationship between IBM Health Checker for z/OS and PFA so that the consumer can take full advantage of the health checker and PFA integration.

PFA is built using remote health checks. Therefore, the health check commands, interface using SDSF, and reporting mechanism available through health checker are fully usable for the PFA checks.

PFA consists of an infrastructure which manages the PFA address space, connects to IBM Health Checker for z/OS (referred to as Health Checker from now on), displays the status of PFA, and launches the JVM™ to model the data to create a prediction.

PFA also contains check-specific code which collects the data for an individual check, models the data to generate a prediction, and compares the actual values to the predictions to issue an exception or an informational message.

It is very important to understand that PFA checks have three basic internal functions. First, they collect data. Data collection for a check is specific to that check meaning that the check's code collects data from the system that is pertinent to the check. For example, if the check needs to calculate a type of storage usage, it interrogates the system control blocks to accumulate the storage used. If it is counting message arrivals of some kind, it uses the appropriate system interface to collect that data. Data collection happens asynchronously on an interval that is configurable by the end user. For example, the default value for the data collection function for checks might be to collect data every 15 minutes. This parameter is called COLLECTINT.

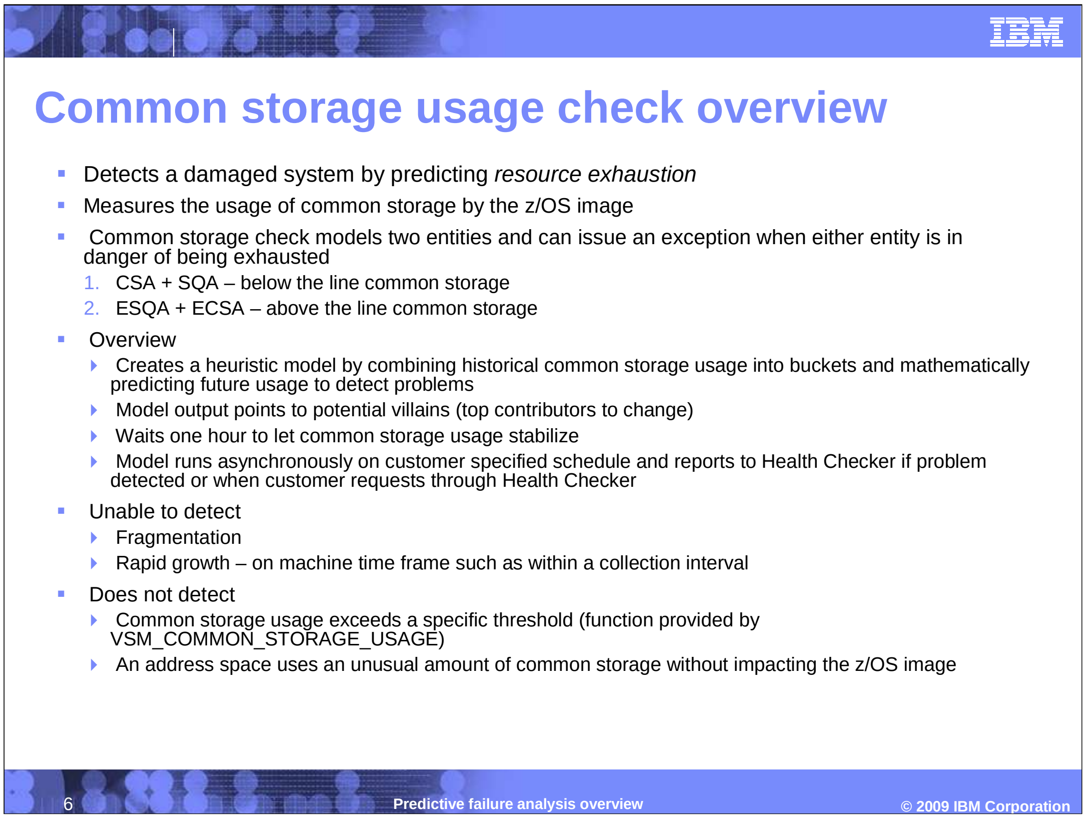
The second major function of PFA checks is to model the data to generate a prediction based on the data collected. This modeling function takes the data that was collected and predicts the value that it expects to see at the end of the model interval or at this point in time. The model interval is also configurable by the end user. For example, the default value for the modeling interval for checks might be to model data every six hours. This parameter is called MODELINT. Modeling also runs asynchronously when it is determined that it is time to model based on the configured value.

The third major function of PFA checks it to perform the comparisons needed to issue an exception or an informational message. It compares what is occurring on the system to what was predicted and issues the appropriate message and report. This function is typically initiated by Health Checker when the time in the INTERVAL parameter for the check is reached. It can also be done for most checks by an end user running the check using Health Checker commands. For some checks, the comparisons are performed at the end of every collection rather than using the INTERVAL parameter in Health Checker.

PFA also manages its data store which is in the UNIX file system. The collected data is stored for use by the modeling code to produce a prediction. The predictions are also stored in the file system for use by the code that performs the comparisons and produces the reports.

All PFA checks have two additional check-specific parameters: COLLECTINACTIVE and DEBUG. The COLLECTINACTIVE parameter is set to yes by default and means to collect and model data for the check even if the check is not ACTIVE(ENABLED) in Health Checker. The DEBUG parameter is used to collect additional debug information to aid in analyzing a PFA problem.

PFA checks can also have check-specific parameters. At this time, each check has a parameter to assist PFA in reducing exception message for situations that will not cause a system outage. These parameters are also configurable by the end user to allow for greater flexibility on a per-system basis.



Common storage usage check overview

- Detects a damaged system by predicting *resource exhaustion*
- Measures the usage of common storage by the z/OS image
- Common storage check models two entities and can issue an exception when either entity is in danger of being exhausted
  1. CSA + SQA – below the line common storage
  2. ESQA + ECSA – above the line common storage
- Overview
  - Creates a heuristic model by combining historical common storage usage into buckets and mathematically predicting future usage to detect problems
  - Model output points to potential villains (top contributors to change)
  - Waits one hour to let common storage usage stabilize
  - Model runs asynchronously on customer specified schedule and reports to Health Checker if problem detected or when customer requests through Health Checker
- Unable to detect
  - Fragmentation
  - Rapid growth – on machine time frame such as within a collection interval
- Does not detect
  - Common storage usage exceeds a specific threshold (function provided by VSM_COMMON_STORAGE_USAGE)
  - An address space uses an unusual amount of common storage without impacting the z/OS image

In the R10 SPE, the PFA team focused on two areas of soft failures. The first was “exhaustion of common resources.” PFA has a check to determine when common storage usage will be exhausted. Note that this check does not monitor individual jobs. It also does not simply detect when a threshold has been reached and alert the system operator. Rather, it uses machine learning and historical data from this particular system to predict the future level of common storage usage and determine if the current trend is going to exceed the available common storage.

The check combines CSA and SQA, and ESQA and ECSA to give predictions for below the line storage and above the line storage.

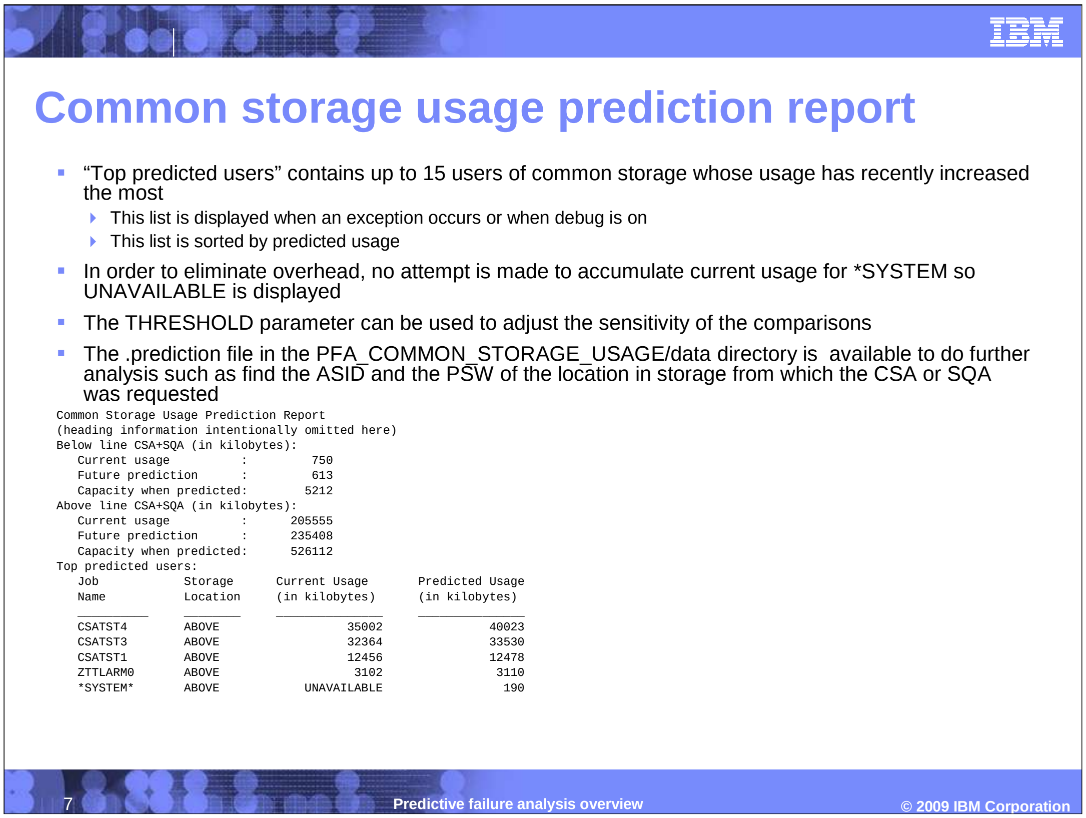
It uses a heuristic model to combine the storage into these two buckets and then mathematically models the future storage to detect problems. If storage is not going to be exhausted, it will issue an informational message. If PFA believes storage will be exhausted before the next model, it will issue an exception. PFA produces a report that shows the top contributors to change. In most cases, the address space causing the storage exhaustion will be in this list.

To reduce false positives after an IPL, the first hour of data collected concerning storage usage is not used in the predictions.

As with all PFA checks, collection and modeling occur asynchronously on a customer-specified schedule.

The common storage usage check is not able to detect all types of common storage exhaustion. It cannot currently detect exhaustion due to fragmentation nor to rapid growth on a machine-time scale.

It is also does not duplicate the function of the VSM_COMMON_STORAGE_USAGE check which detects when common storage usage has exceeded a specific threshold. It also does not detect when an address space uses an unusual amount of common storage without impacting the z/OS image.



Common storage usage prediction report

- "Top predicted users" contains up to 15 users of common storage whose usage has recently increased the most
  - This list is displayed when an exception occurs or when debug is on
  - This list is sorted by predicted usage
- In order to eliminate overhead, no attempt is made to accumulate current usage for *SYSTEM so UNAVAILABLE is displayed
- The THRESHOLD parameter can be used to adjust the sensitivity of the comparisons
- The .prediction file in the PFA_COMMON_STORAGE_USAGE/data directory is available to do further analysis such as find the ASID and the PSW of the location in storage from which the CSA or SQA was requested

```
Common Storage Usage Prediction Report
(heading information intentionally omitted here)
Below line CSA+SQA (in kilobytes):
   Current usage           :         750
   Future prediction       :         613
   Capacity when predicted:         5212
Above line CSA+SQA (in kilobytes):
   Current usage           :      205555
   Future prediction       :      235408
   Capacity when predicted:       526112
Top predicted users:
   Job            Storage     Current Usage       Predicted Usage
   Name           Location    (in kilobytes)      (in kilobytes)
   __________     ________    _______________     _______________
   CSATST4        ABOVE                 35002               40023
   CSATST3        ABOVE                 32364               33530
   CSATST1        ABOVE                 12456               12478
   ZTTLARM0       ABOVE                  3102                3110
   *SYSTEM*       ABOVE           UNAVAILABLE                 190
```

A partial Common Storage Usage check prediction report is shown on this chart. Notice that the "below the line" and the "above the line" storage are represented separately. They are also compared separately and either can cause an exception.

A common heading is provided on the reports of all PFA checks that will provide the last data collection time, the collection interval, the last model time, and the model interval.

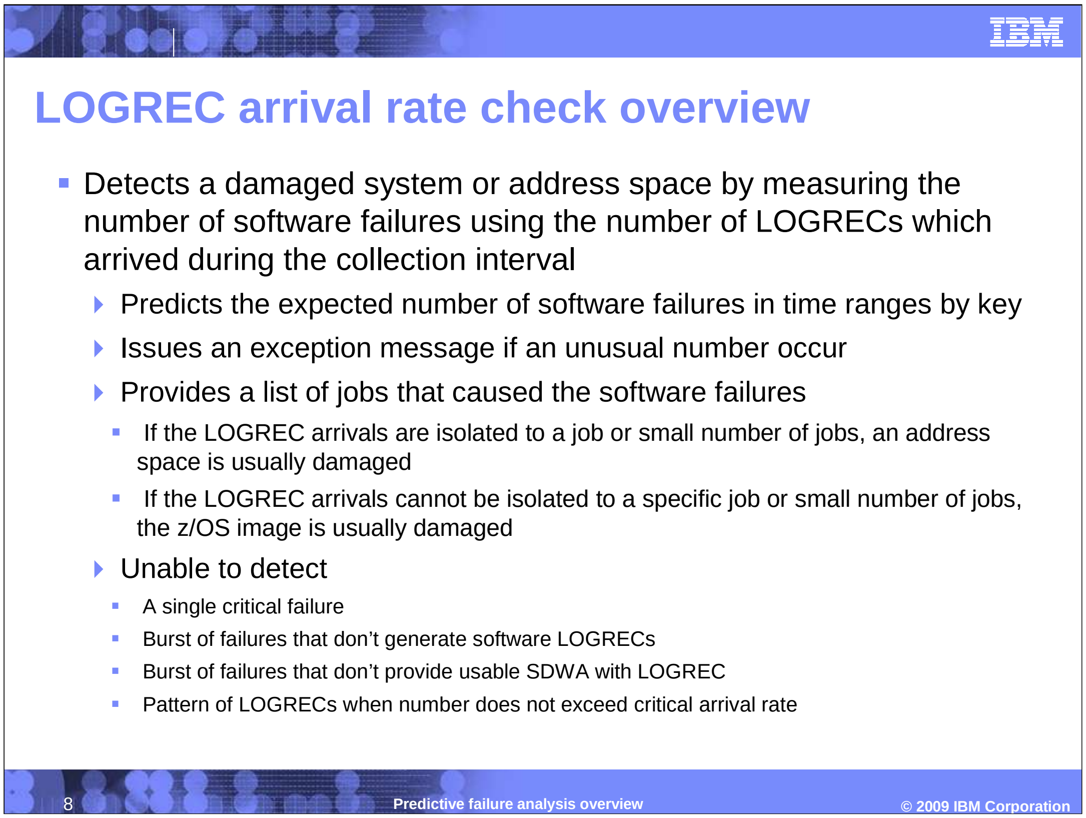
All values are in kilobytes.

The future prediction is the value that was modeled at the last model interval that is the predicted value at the end of the model interval (or two hours ahead if the model interval is less than two hours).

The top predicted users' list is provided only if an exception is issued or debug is on. If the check determines that there is no problem such that the informational message is issued, this list is not provided. Producing this list requires some performance overhead and due to the fact that this check's comparisons are done every minute, the list of jobs is not provided unless necessary to analyze a problem.

The top predicted users' list can be used to determine the address spaces causing the potential storage exhaustion. If more information is needed, the files in this check's /data directory can be used for further analysis. Details on what information is provided in each file can be found in the PFA documentation.

Due to the fact that the *SYSTEM* owned storage can be from many places, the current usage is not accumulated on the report to further reduce overhead.

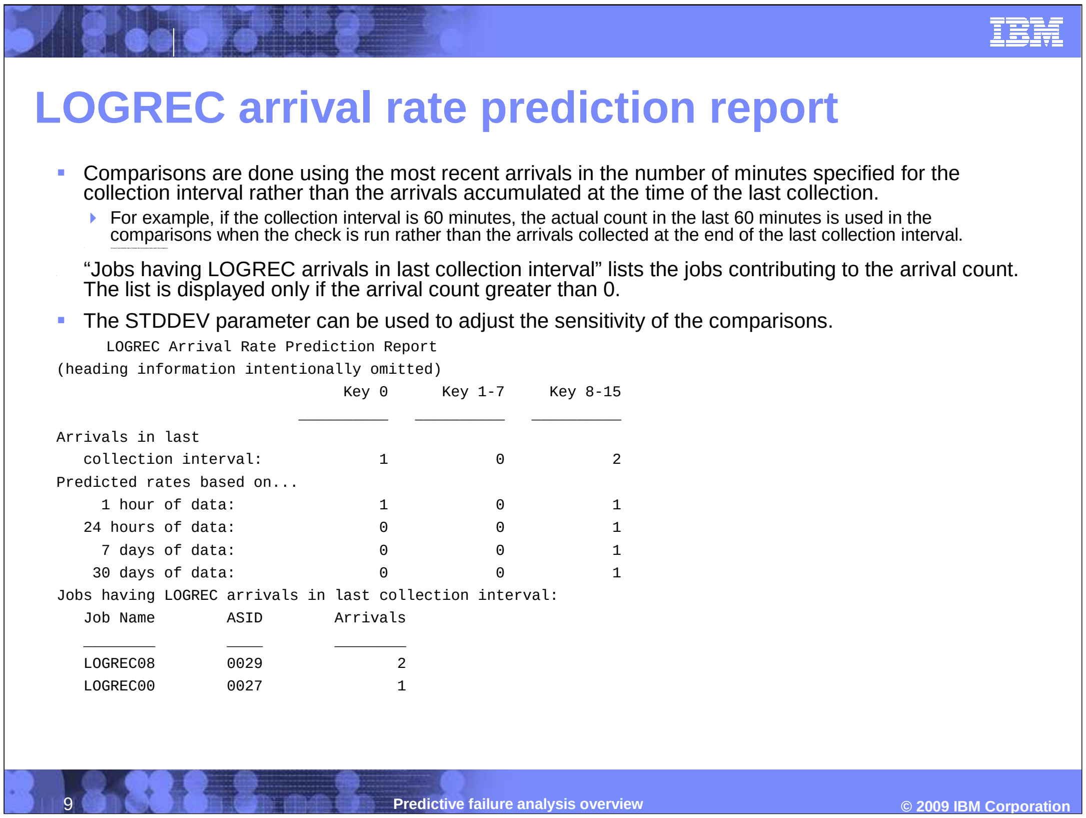


LOGREC arrival rate check overview

- Detects a damaged system or address space by measuring the number of software failures using the number of LOGRECs which arrived during the collection interval
  - Predicts the expected number of software failures in time ranges by key
  - Issues an exception message if an unusual number occur
  - Provides a list of jobs that caused the software failures
    - If the LOGREC arrivals are isolated to a job or small number of jobs, an address space is usually damaged
    - If the LOGREC arrivals cannot be isolated to a specific job or small number of jobs, the z/OS image is usually damaged
  - Unable to detect
    - A single critical failure
    - Burst of failures that don’t generate software LOGRECs
    - Burst of failures that don’t provide usable SDWA with LOGREC
    - Pattern of LOGRECs when number does not exceed critical arrival rate

The second type of soft failure which PFA detects in R10 is for excessive failures by key. This check is called the LOGREC Arrival Rate check. It measures the number of software failures using the number of LOGRECs which arrive during the collection interval. It categorizes the software failures by key into three categories. The three categories are key 0, keys 1-8, and keys 9-15. Each category can trigger the exception for this check because PFA creates predictions for the expected number of software failures for each of these key categories. If the number of exceptions is unusually high, the exception is issued. A list of jobs that had high arrivals are included in the report. If the arrivals are isolated to a single job or a small number of jobs, the address space or address spaces are damaged. If the arrivals cannot be isolated, the z/OS image is likely damaged.

The LOGREC arrival rate check is not able to detect all types of failures that could lead to a system soft failure. It cannot detect a single critical failure. It also does not track other types of failures. It only tracks software failures. It needs the SDWA record with the LOGREC in order to track the failure. Therefore, if there is no usable SDWA record for the failure, it will not be tracked by the check. It also is looking for an excessive number of LOGRECs. Therefore, if there is a pattern of LOGRECs where the number of LOGRECs arriving does not exceed the critical rate, no exception can be issued.



LOGREC arrival rate prediction report

- Comparisons are done using the most recent arrivals in the number of minutes specified for the collection interval rather than the arrivals accumulated at the time of the last collection.
  - For example, if the collection interval is 60 minutes, the actual count in the last 60 minutes is used in the comparisons when the check is run rather than the arrivals collected at the end of the last collection interval.

“Jobs having LOGREC arrivals in last collection interval” lists the jobs contributing to the arrival count. The list is displayed only if the arrival count greater than 0.

- The STDDEV parameter can be used to adjust the sensitivity of the comparisons.

```
      LOGREC Arrival Rate Prediction Report
(heading information intentionally omitted)
                                Key 0      Key 1-7     Key 8-15
                           __________  __________  __________

Arrivals in last
   collection interval:            1           0           2
Predicted rates based on...
     1 hour of data:               1           0           1
   24 hours of data:               0           0           1
     7 days of data:               0           0           1
    30 days of data:               0           0           1
Jobs having LOGREC arrivals in last collection interval:
   Job Name        ASID        Arrivals

   ________        ____        ________
   LOGREC08        0029               2
   LOGREC00        0027               1
```

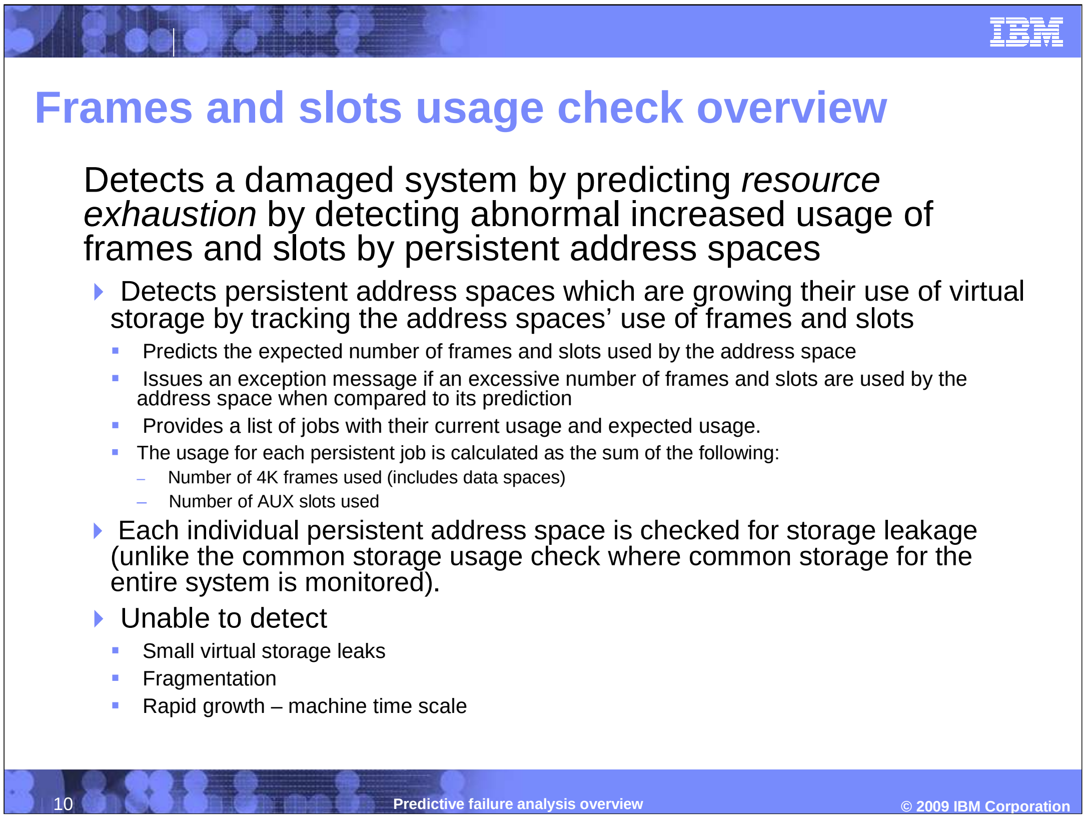
This chart shows the LOGREC arrival rate prediction report.

It is important to understand what timeframes the arrivals occurred to understand which arrivals are included in the comparison. The comparisons are done using the most recently arrived LOGRECs compared to the modeled predictions for the three key buckets and for the four time ranges which will be described in more detail. The most recent arrivals are accumulated from the current time back for the number of minutes specified in the collection interval. The report shows the most recent arrivals as the “Arrivals in last collection interval.”

PFA models the LOGREC arrivals for three buckets of keys – key 0, keys 1-7, and keys 8-15. It models predictions for those three categories for four different ranges of data. That is, the arrivals are modeled using data that was collected in the last hour, data that was collected over the last 24 hours of data, data that was collected over the last 7 days of data, and data that was collected over the last 30 days of data. If there is not enough data collected for all of those time ranges, the line for that time is not included on the report.

The report shows the arrivals in the last collection interval’s worth of time by key and the predictions by key for predictions based on those four different models of data. Any one of those keys and prediction times can cause the exception to be issued when compared to the arrivals in the last collection interval’s worth of time after the configurable standard deviation is applied.

The list of jobs in the “Jobs having LOGREC arrivals in last collection interval” list the jobs that contributed to the arrival count. If there were no arrivals, the list is not displayed. Most often, the job or jobs with the most arrivals are the reason the exception was issued.



Frames and slots usage check overview

Detects a damaged system by predicting *resource exhaustion* by detecting abnormal increased usage of frames and slots by persistent address spaces

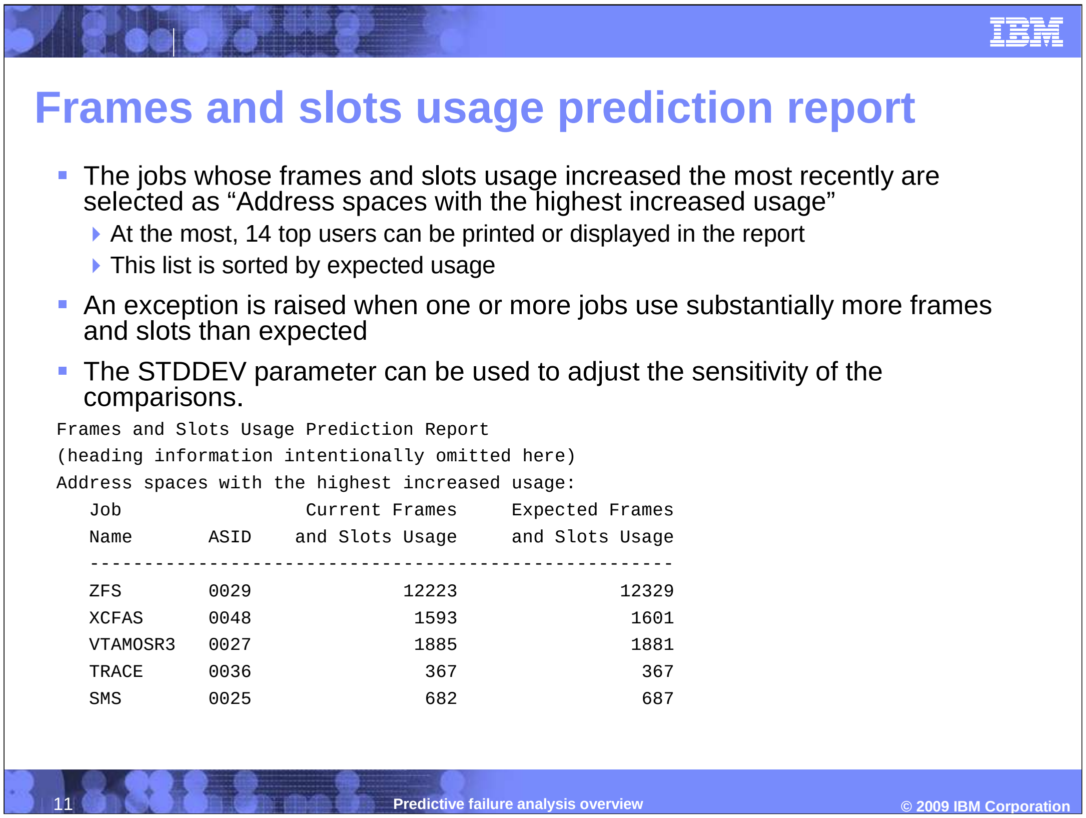
- Detects persistent address spaces which are growing their use of virtual storage by tracking the address spaces’ use of frames and slots
  - Predicts the expected number of frames and slots used by the address space
  - Issues an exception message if an excessive number of frames and slots are used by the address space when compared to its prediction
  - Provides a list of jobs with their current usage and expected usage.
  - The usage for each persistent job is calculated as the sum of the following:
    - Number of 4K frames used (includes data spaces)
    - Number of AUX slots used
- Each individual persistent address space is checked for storage leakage (unlike the common storage usage check where common storage for the entire system is monitored).
- Unable to detect
  - Small virtual storage leaks
  - Fragmentation
  - Rapid growth – machine time scale

10 Predictive failure analysis overview © 2009 IBM Corporation

The first check available starting with R11 also detects exhaustion of shared resources. The Frames and Slots Usage check was created to detect virtual storage leaks in a persistent address space by detecting an abnormally high usage of frames and slots when compared to the expected usage. A “persistent” address space is defined to be an address space that starts within one hour after IPL.

The frames and slots usage check tracks frames and slots used by persistent address spaces. It models a prediction for the persistent address spaces that have had the largest change in their frames and slots storage usage. It predicts the number of frames and slots that the address space is expected to be consuming. It issues an exception if one of these persistent address spaces exceeds the prediction for that address space after the standard deviation is applied. The report issued with the exception lists the address spaces whose current usage is abnormally high.

The frames and slots usage check is not able to detect small virtual storage leaks nor is it meant as a virtual storage monitor. It cannot detect virtual storage fragmentation nor can it detect rapid growth on a machine-time scale.



Frames and slots usage prediction report

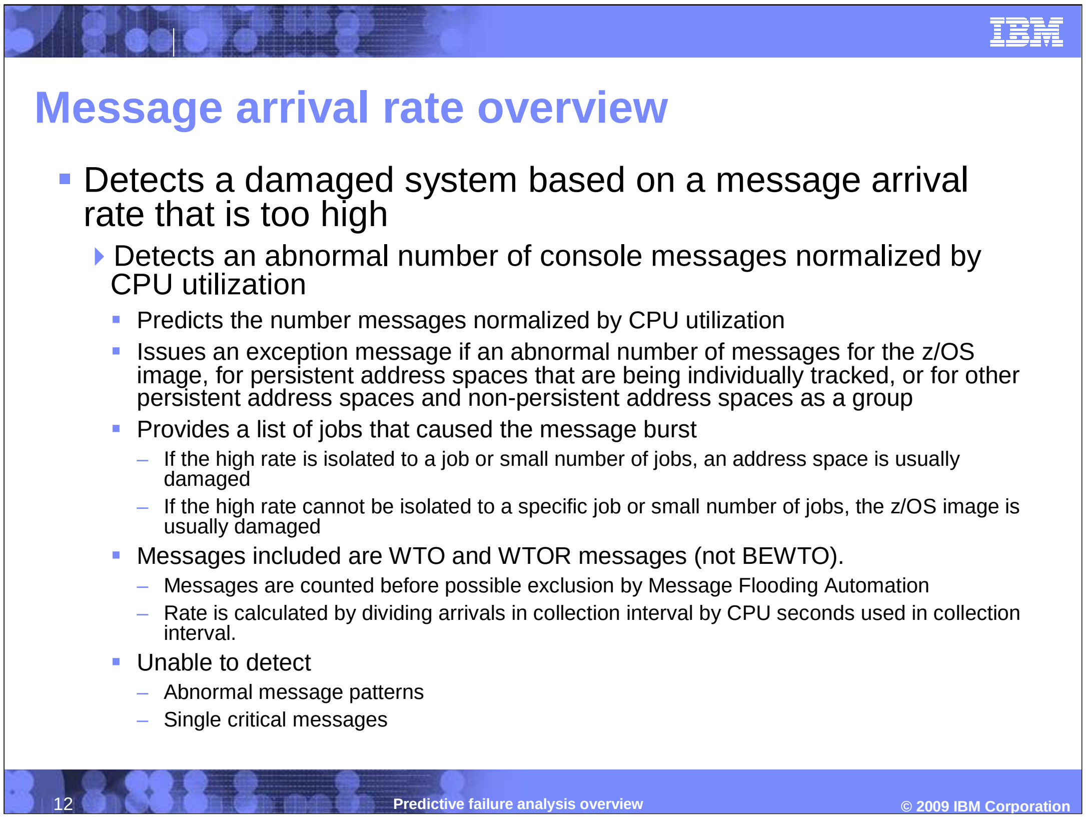
- The jobs whose frames and slots usage increased the most recently are selected as “Address spaces with the highest increased usage”
  - At the most, 14 top users can be printed or displayed in the report
  - This list is sorted by expected usage
- An exception is raised when one or more jobs use substantially more frames and slots than expected
- The STDDEV parameter can be used to adjust the sensitivity of the comparisons.

```
Frames and Slots Usage Prediction Report
(heading information intentionally omitted here)
Address spaces with the highest increased usage:
   Job                Current Frames     Expected Frames
   Name       ASID   and Slots Usage     and Slots Usage
   ------------------------------------------------------
   ZFS        0029             12223               12329
   XCFAS      0048              1593                1601
   VTAMOSR3   0027              1885                1881
   TRACE      0036               367                 367
   SMS        0025               682                 687
```

The address spaces with the highest increased usage are those whose usage has recently increased the most. Only the top 14 persistent jobs are selected to be printed or displayed in the report when no exception is issued. This list is sorted by expected usage. If an exception is issued, only those jobs causing the exception are listed.

The report will show the current and the expected usage for the jobs displayed. The current and expected usage values shown are the total number of 4K frames and AUX slots in used.

Once the modeling is done, the current usage is compared to the expected usage for each individual persistent address space and the value in the user-configurable STDDEV parameter is applied. If it is determined that an address space is using substantially more frames and slots compared to the expected number, an exception is issued. This algorithm differs from the CSA check where an exception is raised for the entire storage usage, not for individual persistent address spaces.



Message arrival rate overview

- Detects a damaged system based on a message arrival rate that is too high
  - Detects an abnormal number of console messages normalized by CPU utilization
    - Predicts the number messages normalized by CPU utilization
    - Issues an exception message if an abnormal number of messages for the z/OS image, for persistent address spaces that are being individually tracked, or for other persistent address spaces and non-persistent address spaces as a group
    - Provides a list of jobs that caused the message burst
      - If the high rate is isolated to a job or small number of jobs, an address space is usually damaged
      - If the high rate cannot be isolated to a specific job or small number of jobs, the z/OS image is usually damaged
    - Messages included are WTO and WTOR messages (not BEWTO).
      - Messages are counted before possible exclusion by Message Flooding Automation
      - Rate is calculated by dividing arrivals in collection interval by CPU seconds used in collection interval.
    - Unable to detect
      - Abnormal message patterns
      - Single critical messages

Predictive failure analysis overview

The second check available in R11 detects excessive failures. The Message Arrival Rate check detects an abnormal arrival rate of console messages. That is, it counts the number of write to operator (WTO) and write to operator reply (WTOR) messages. It specifically excludes branch entry write to operator (BEWTO) messages. The rate is calculated by dividing the count of the arrivals in the collection interval with the number of seconds of CPU used in the collection interval.

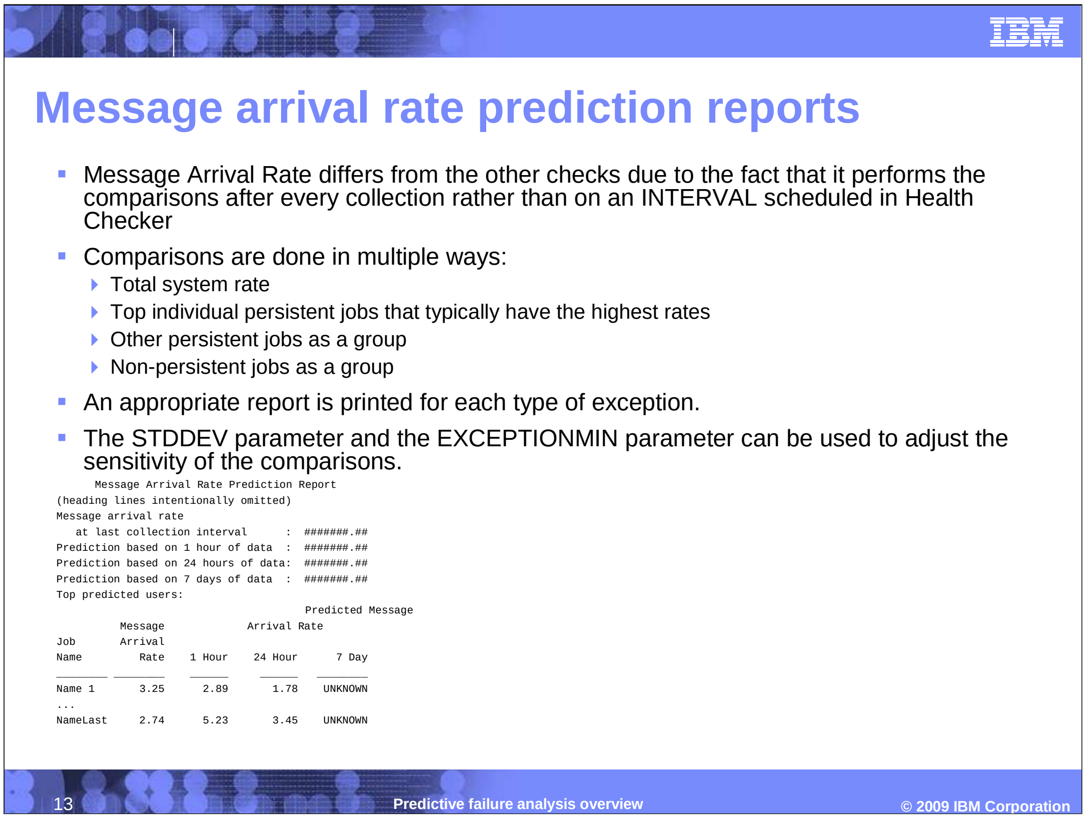
If the arrival rate found at the last collection is excessively high when compared to the prediction, an exception message is issued. The exception message can be issued by comparing the collected and predicted rates for the entire system, for each individual persistent job being tracked, for the other persistent jobs as a group, or for the non-persistent jobs as a group.

The definition of persistent jobs is the same as for the frames and slots usage check. That is, the job is considered persistent if it starts within one hour after IPL. All message arrival rates collected in the first hour after IPL are discarded to allow the system time to stabilize after the IPL.

The persistent jobs that are tracked individually are determined either by the jobs that were tracked before IPL or the jobs that had the highest arrivals after a six hour warm-up phase that begins an hour after IPL or when PFA starts, whichever is later. If PFA had not previously been running or data had not been collected before the IPL, PFA chooses the individual persistent jobs to track based on the arrival rates in the 6 hours from hour 1 to hour 7 after IPL. The persistent jobs with the highest arrival rates are tracked individually. All other persistent jobs are put in the “other persistent jobs” category. If PFA had been running prior to IPL and had been collecting data, the jobs that were previously tracked are tracked again if that entire list of jobs is persistent again after the IPL. And, if PFA had collected data prior to IPL, the last hour’s worth of data collected that exists in the files when PFA starts again is discarded so that the arrivals collected during shutdown do not skew the predictions.

Messages are counted before possible exclusion by message flood automation.

The message arrival rate check is not designed to detect abnormal message patterns or single critical messages.



Message arrival rate prediction reports

- Message Arrival Rate differs from the other checks due to the fact that it performs the comparisons after every collection rather than on an INTERVAL scheduled in Health Checker
- Comparisons are done in multiple ways:
  - Total system rate
  - Top individual persistent jobs that typically have the highest rates
  - Other persistent jobs as a group
  - Non-persistent jobs as a group
- An appropriate report is printed for each type of exception.
- The STDDEV parameter and the EXCEPTIONMIN parameter can be used to adjust the sensitivity of the comparisons.

```
      Message Arrival Rate Prediction Report
(heading lines intentionally omitted)
Message arrival rate
   at last collection interval      :  #######.##
Prediction based on 1 hour of data  :  #######.##
Prediction based on 24 hours of data:  #######.##
Prediction based on 7 days of data  :  #######.##
Top predicted users:
                                       Predicted Message
          Message             Arrival Rate
Job       Arrival
Name         Rate    1 Hour    24 Hour     7 Day
________ ________    ______     ______   ________
Name 1       3.25      2.89       1.78   UNKNOWN
...
NameLast     2.74      5.23       3.45   UNKNOWN
```

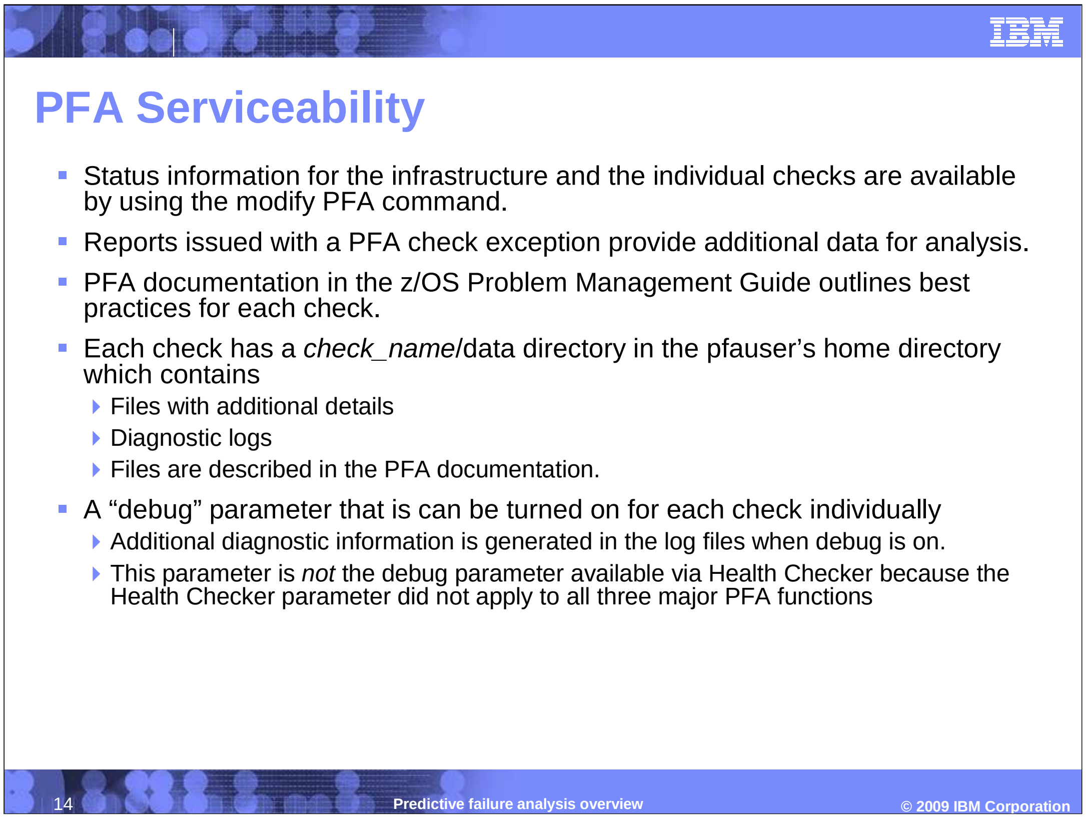
The message arrival rate check differs from the other checks in the way the check is run to do the comparisons. Rather than having a set time INTERVAL as a Health Checker parameter, the check is designed to run after every collection. By performing the check automatically upon successful completion of a collection, the check is able to compare the most recent arrivals with the predictions modeled at the last model interval. This enhances both the performance of the check itself as well as the responsiveness of the check to the current activity of the system.

The message arrival rate check performs the comparison in several ways. It can detect a damaged system based on the total arrival rate of the system. It can also detect a damaged address space that might lead to a damaged system. It tracks persistent jobs that had the highest arrival rate during an internal warm-up period or that were the same jobs being tracked before IPL if those jobs are still available and the check had data from prior to IPL. If any of those persistent jobs have higher rates than expected, an exception will be issued. It also tracks the other persistent jobs as a group and an exception can be issued due to those. It also tracks the non-persistent jobs as a group and an exception can be issued due to those.

The message arrival rate check is similar to the LOGREC arrival rate check in the aspect that they both make predictions based on 1 hour of data, 24 hours of data, and 7 days of data. Message arrival rate does not model back using 30 days of data, however. If the amount of data required for any comparison is not available for the system as a whole, that line is not printed on the report. For the jobs listed in the report details, if not enough data is not available for a particular job for any given timeframe, UNKNOWN is printed on the report.

The STDDEV parameter and the EXCEPTIONMIN parameter can be used to adjust the sensitivity of the comparisons. For example, if an exception is issued for an address space whose current arrival rate or predicted arrival rate are quite low and deemed insignificant, the EXCEPTIONMIN parameter can be increased above those values to avoid the exception. The STDDEV parameter can also be increased or reduced to further improve the calculations to fit your specific needs.

The TRACKEDMIN parameter can be used to avoid having persistent address spaces with low rates be tracked at the first startup of PFA. Currently, the parameter has the default of 0 so that all address spaces are candidates to be tracked.



PFA Serviceability

- Status information for the infrastructure and the individual checks are available by using the modify PFA command.
- Reports issued with a PFA check exception provide additional data for analysis.
- PFA documentation in the z/OS Problem Management Guide outlines best practices for each check.
- Each check has a *check_name*/data directory in the pfauser’s home directory which contains
  - Files with additional details
  - Diagnostic logs
  - Files are described in the PFA documentation.
- A “debug” parameter that is can be turned on for each check individually
  - Additional diagnostic information is generated in the log files when debug is on.
  - This parameter is *not* the debug parameter available via Health Checker because the Health Checker parameter did not apply to all three major PFA functions

Originally, PFA was designed to be a black box. Customers were just supposed to start it and forget it. However, it was later decided through customer interaction that more information was needed in order to ensure that PFA was running correctly. Therefore, a comprehensive “modify PFA” command was built to display status information for the PFA infrastructure as well as detailed status for each individual check.

When a check exception is issued, the reports previously described should greatly assist in analyzing the problem.

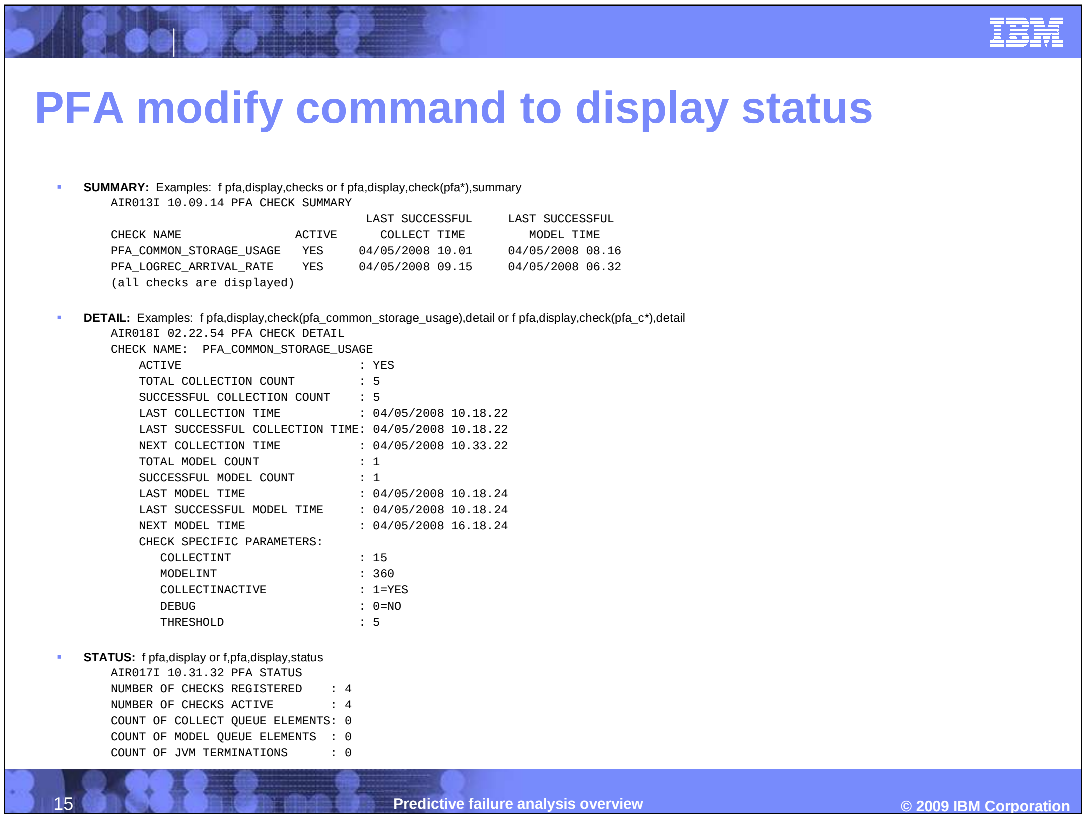
In addition, the PFA documentation outlines best practices for each check to provide advice on how to analyze the problem.

Each check has a /data directory in the pfauser’s home directory. For example, if the pfa user is “pfauser,” the data directory for the CSA usage check would be /u/pfauser/PFA_COMMON_STORAGE_USAGE/data. The files needed by PFA to collect data, model data, and perform the check are found in the /data directory. These files are documented in the PFA documentation and some can be used to help you analyze the exception data.

If a PFA problem is suspected, the /data directory also contains log files that contain additional debug information. If the PFA debug parameter is on, additional information will be found in the log files.

If a PFA problem is suspected, IBM service will likely request the last exception report and the /data for the check. It is preferred that the problem has been re-created with debug on.

Note that the debug parameter for PFA is not the debug parameter provided by Health Checker. Rather, it is a parameter listed in the check-specific parameters for each check. The Health Checker debug parameter was not able to be used because it could not be applied to all operations within PFA such as collect, model, and perform comparison operations.



PFA modify command to display status

- **SUMMARY:** Examples: f pfa,display,checks or f pfa,display,check(pfa*),summary

```
AIR013I 10.09.14 PFA CHECK SUMMARY
                                    LAST SUCCESSFUL     LAST SUCCESSFUL
CHECK NAME                ACTIVE      COLLECT TIME         MODEL TIME
PFA_COMMON_STORAGE_USAGE   YES     04/05/2008 10.01     04/05/2008 08.16
PFA_LOGREC_ARRIVAL_RATE    YES     04/05/2008 09.15     04/05/2008 06.32
(all checks are displayed)
```

- **DETAIL:** Examples: f pfa,display,check(pfa_common_storage_usage),detail or f pfa,display,check(pfa_c*),detail

```
AIR018I 02.22.54 PFA CHECK DETAIL
CHECK NAME:  PFA_COMMON_STORAGE_USAGE
    ACTIVE                         : YES
    TOTAL COLLECTION COUNT         : 5
    SUCCESSFUL COLLECTION COUNT    : 5
    LAST COLLECTION TIME           : 04/05/2008 10.18.22
    LAST SUCCESSFUL COLLECTION TIME: 04/05/2008 10.18.22
    NEXT COLLECTION TIME           : 04/05/2008 10.33.22
    TOTAL MODEL COUNT              : 1
    SUCCESSFUL MODEL COUNT         : 1
    LAST MODEL TIME                : 04/05/2008 10.18.24
    LAST SUCCESSFUL MODEL TIME     : 04/05/2008 10.18.24
    NEXT MODEL TIME                : 04/05/2008 16.18.24
    CHECK SPECIFIC PARAMETERS:
       COLLECTINT                  : 15
       MODELINT                    : 360
       COLLECTINACTIVE             : 1=YES
       DEBUG                       : 0=NO
       THRESHOLD                   : 5
```

- **STATUS:** f pfa,display or f,pfa,display,status

```
AIR017I 10.31.32 PFA STATUS
NUMBER OF CHECKS REGISTERED    : 4
NUMBER OF CHECKS ACTIVE        : 4
COUNT OF COLLECT QUEUE ELEMENTS: 0
COUNT OF MODEL QUEUE ELEMENTS  : 0
COUNT OF JVM TERMINATIONS      : 0
```

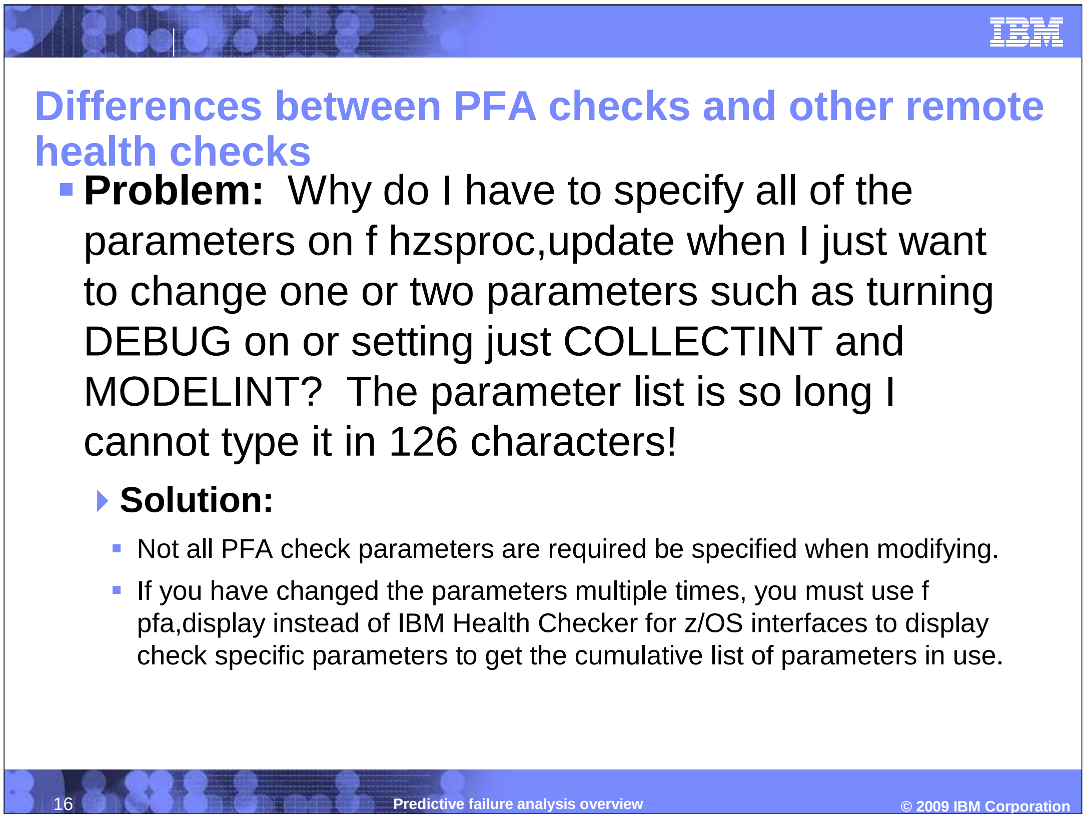
This chart shows examples of the PFA modify command. It can be used to display summary or detailed information for the PFA checks and to display status information for the PFA infrastructure.

The syntax of the PFA modify command is very similar to the Health Checker modify command and is documented clearly in the PFA documentation.

Summary information for the checks shows the check name, whether it is active in health checker, the last collection time and the last model time. Either all checks can be shown or individual checks can be shown by specifying the name of a check or a wildcard that matches more than one check. Wildcards can be specified as the last character of the check name.

Detailed information for a check shows counts and times for collection and modeling. It also shows the parameters for specific to this check. In fact, in order to display the cumulative set of parameters for this check, the PFA modify command to display must be used. PFA allows the user to modify parameters individually and accumulate the changes rather than requiring all parameters to be specified when modifying. Displaying the check’s parameters with Health checker commands will not show the cumulative list of parameters if the parameters were changed using more than one modify command.

The PFA infrastructure status can also be displayed using the PFA modify command. The number of checks registered is the number of checks that exist to the PFA infrastructure. The number of checks active are the number of PFA checks that are ACTIVE(ENABLED) in Health Checker.

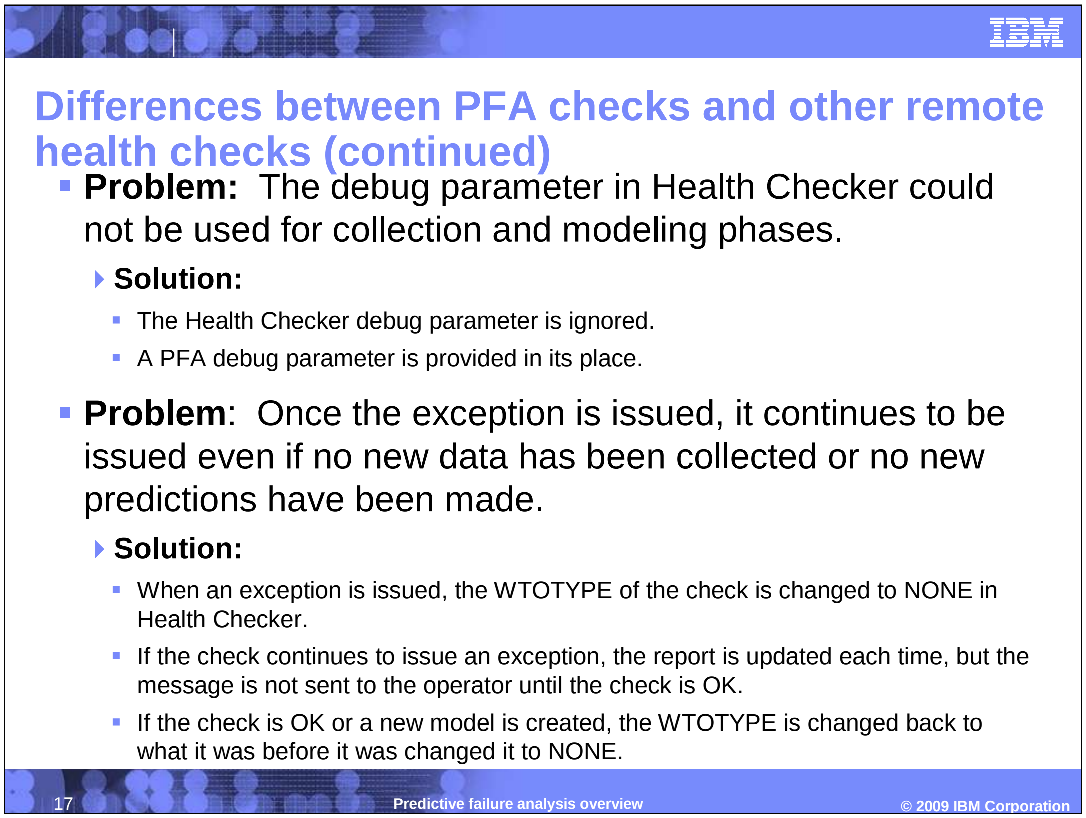


Differences between PFA checks and other remote health checks

- **Problem:** Why do I have to specify all of the parameters on f hzsproc,update when I just want to change one or two parameters such as turning DEBUG on or setting just COLLECTINT and MODELINT? The parameter list is so long I cannot type it in 126 characters!
  - **Solution:**
    - Not all PFA check parameters are required be specified when modifying.
    - If you have changed the parameters multiple times, you must use f pfa,display instead of IBM Health Checker for z/OS interfaces to display check specific parameters to get the cumulative list of parameters in use.

The PFA checks have several parameters. Health Checker requires all parameters to be specified on a modify even if only one parameter is being changed. And, if using the Health Checker modify command on the command line, only 126 characters are allowed and not all PFA check parameters could be specified. In addition, it was not considered very usable to need to input all parameters just to change the value of one parameter.

Therefore, PFA made a change so that not all parameters need to be specified when modifying parameters using Health Checker. PFA internally tracks the values of each parameter so that if not all parameters are specified, the previous value for the parameters not specified are retained. However, Health Checker does not know about the internal storage of the PFA parameters. Therefore, it only has the capability of displaying the last modify operation performed. If you use multiple modify operations or do not specify all of the parameters, Health Checker can only display the last parameter modified. Therefore, in order to see all parameters in use by any PFA check, the modify PFA display command must be used.

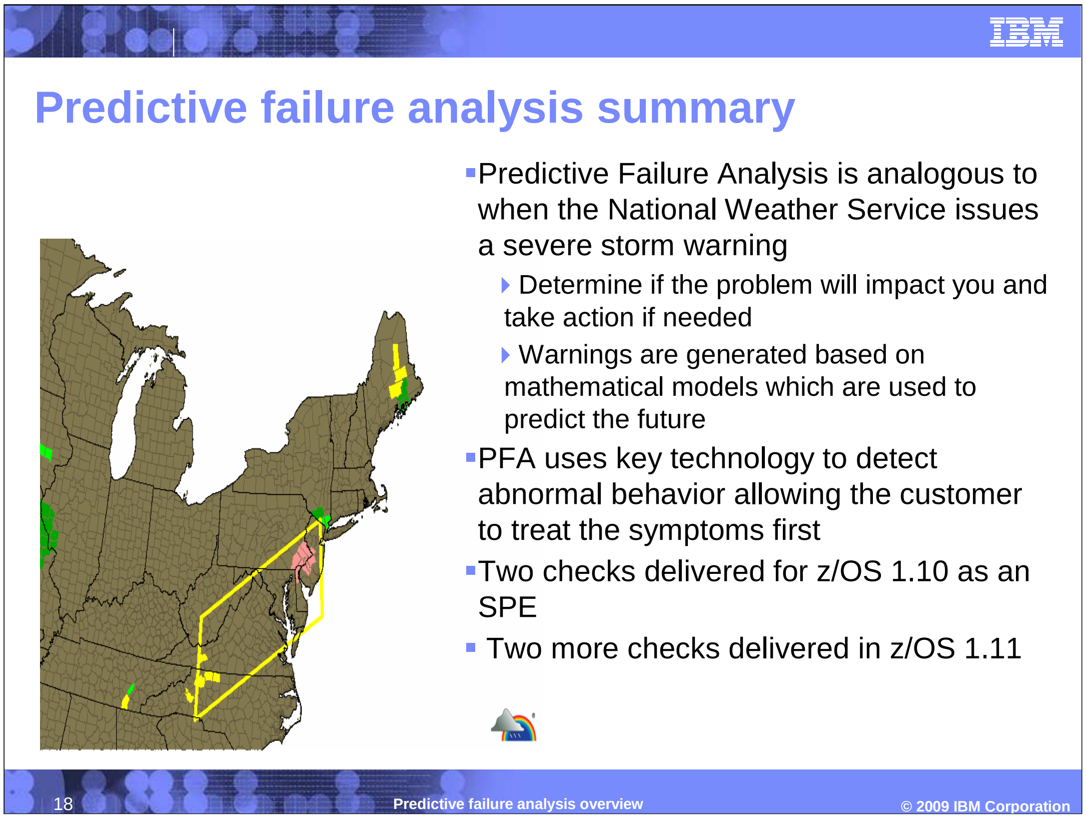


Differences between PFA checks and other remote health checks (continued)

- **Problem:** The debug parameter in Health Checker could not be used for collection and modeling phases.
  - **Solution:**
    - The Health Checker debug parameter is ignored.
    - A PFA debug parameter is provided in its place.
- **Problem**: Once the exception is issued, it continues to be issued even if no new data has been collected or no new predictions have been made.
  - **Solution:**
    - When an exception is issued, the WTOTYPE of the check is changed to NONE in Health Checker.
    - If the check continues to issue an exception, the report is updated each time, but the message is not sent to the operator until the check is OK.
    - If the check is OK or a new model is created, the WTOTYPE is changed back to what it was before it was changed it to NONE.

When the debug parameter in Health Checker was set, PFA was not notified until the next run of the check. However, debug data needed to be generated for the collect and model phases of the checks as well. Therefore, every PFA check has a debug parameter that is a check-specific parameter which applies to all phases of PFA checks. The debug parameter in Health Checker is ignored by PFA.

Once an exception was issued, the exception was issued every time the check was run even though new data had not yet been collected and no new predictions had been modeled. This problem was especially annoying for system operators that carried a pager! Therefore, PFA was changed so that when an exception is issued, the check’s WTOTYPE is changed to NONE in Health Checker. The check continues to collect data, model predictions, and perform comparisons. However, if the check continues to issue an exception, the report is updated with the most recent information, but the message is not sent as a write to operator message. Once a new model is created or when the check is run without issuing an exception, the WTOTYPE is changed back to what it was before it was changed to NONE.



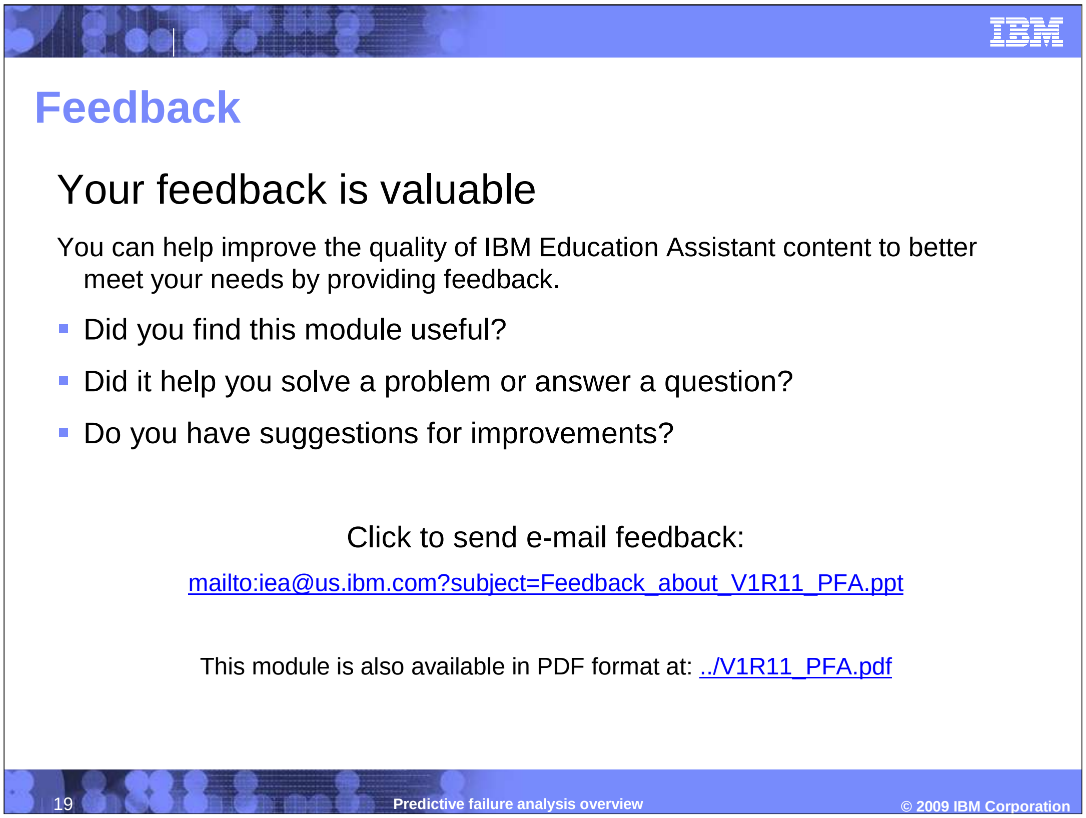
Predictive failure analysis summary

- Predictive Failure Analysis is analogous to when the National Weather Service issues a severe storm warning
  - Determine if the problem will impact you and take action if needed
  - Warnings are generated based on mathematical models which are used to predict the future
- PFA uses key technology to detect abnormal behavior allowing the customer to treat the symptoms first
- Two checks delivered for z/OS 1.10 as an SPE
- Two more checks delivered in z/OS 1.11

Predictive failure analysis uses technology which is new to detect when abnormal behavior is occurring on the system to allow the operations team to treat the symptoms of the problem before they fully understand the cause. It is analogous to when the National Weather Service issues a severe storm warning. Data is provided to help determine the cause of the problem and best practices are documented to provide guidance to the solution.

Predictive failure analysis is using mathematical techniques borrowed from other industries that determine if jet engines will fail or if nuclear power plants need to be shutdown.

The common storage usage check and the LOGREC arrival rate check were delivered as an SPE for R10. The frames and slots usage check and the message arrival rate check were delivered in R11.



Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_V1R11_PFA.ppt

This module is also available in PDF format at: ../V1R11_PFA.pdf

19 Predictive failure analysis overview

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, copyrights, and disclaimers

IBM, the IBM logo, ibm.com, and the following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

CICS Current z/OS

If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of other IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

JVM, ZFS, and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.