GEORG AUGUST UNIVERSITÄT GÖTTINGEN
FACULTY OF BUSINESS AND ECONOMICS, APPLIED STATISTICS

Statistical internship as part of the Master's program Sept 21, 2024

Modelling of System Failure Statistics based on Node Components

by

Emmanuel TCHOUMKEU NGATAT

Abstract: When it comes to failures, one is tempted to give reasons for their occurence, but only after they have already happened. In this work, i put myself before the failure occurrence, making great use of historical data to model, and even predict failure probabilities of HPC system components of the GWDG. The failures are of different types and the system has a structure which is constructed in parallel. This implies multiple nodes/clusters, and therefore a flexible model for explaining and aggregating results. As i do not predict failures themselves but rather am concerned with the proper modelling of their rates, i shall show that my approach combining time series analysis, mixed models and Markov chains have a great power and efficiency even in case of Big Data, while offering a very fast implementation time in R.

Msc. Jonathan Decker and Prof.Dr. Julian Kunkel were my supervisors. Dr. Michael Bidollahkhani was a great help in the data generating process. Prof.Dr. Thomas Kneib is the statistical advisor, and the end examiner. I thank them for their disposition every time. I didn't use any type of KI-System or Chat-GPT. Every source is cited accordingly.

Keywords: *Time series, Markov chain, failure rate, MTBF, HPC, GWDG, Big Data, Node, Cluster, ICC*.

Contents

1 Introduction

Hard- and Software components does not significantly differ in the new generation of computers. Architectures commonly only differs in the number/type of cpus and/or gpus included as well as the repartition of tasks over the nodes and their priorities. Nodes can not be assumed to be equal, for they don't provide the same possibilities in terms of capacity, performance and efficiency. In this work i will show that it is possible to take advantage of historical data to preserve system integrity, avoid false node selection based on the type of job one would like to perform. Therefore i will first present the automated efficient functions i created to collect and pre-process raw data efficiently in R, then i will explain the MTBF (Mean Time Between Failures) mechanism as efficient measure to model failure probabilities for clustered time series data. Based on the so first gathered cleaned data, I will perform some statistical exploratory data analysis, an MCMC algorithm to estimate failure distributions and will then conclude, if time is enough with an application of the ICC (Intraclass correlation coefficient) for time series forecasting.

2 Collection and Preprocessing

The real size of the data i am working with is about 500 GB. A typical machine learning workflow involves data collection, pre-processing, dataset building, model training, evaluation, deployment, and continuous monitoring and updating. It is pre-configured and may require revisiting previous steps based on the results at later stages. It is therefore possible to get so called „log-files“ from a HPC system machine, which are a documentation of the main processes and reports. The log-files were compressed and given to my disposition under https://owncloud.gwdg.de/index.php/s/P4Y9xelYAprGZBG. Decompressing the downloaded data would take too much time.

The log-files concerns three different nodes labelled „c0111“, „gcn1111“ and „gcn2222“ over one year from approx. April 2023 until April 2024. I have no clue what is the main difference between them at the project start. The log-files are composed of many files in *-gz* format. For each node, two types of data are provided:

- files with a label beginning with ”slurm-” are informations about the node statuses. They contain measurements on a particular node taken from a

sensor two time per hour (e.g. temperature, performance [watts], energy [joules]);

- and files with a label beginning with ”syslog-” contain error rapports, that are recorded on a continuous basis.

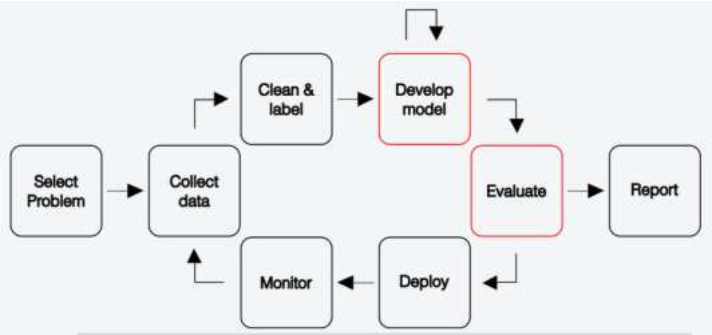
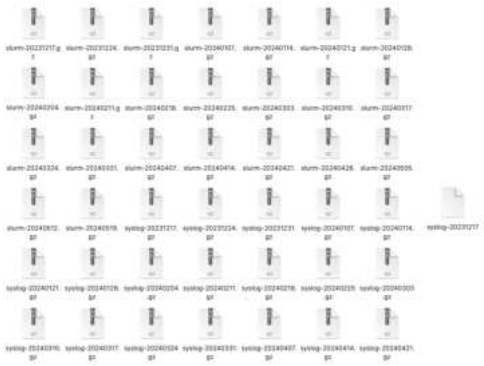


Figure 1: Machine Learning Workflow to produce a log-report
Source: *https://info.gwdg.de/news/best-practices-for-machine-learning-with-hpc/*

I am only interested in the latest, for the measurements do not assume equidistant time knots [1].



Out of this raw data, my aim is first to construct a simple data frame with three different columns (*Date*, *Message* and *Node*), which i am going to pre-process for

[1] If it was the case, so each time stamp would be separated from the latest with the same time amount and the calculation of MTBF for the modelling of elapsed times between failures would then be biased.

analysis purposes. In the following i describe the data gathering process threw the ***get_data_from_raw*** function in R. I arrive to challenge the big amount a data by a single but determinant R-command (*readLines(log_file)*), so i must not decompress even a single file to read. The entire execution for the three nodes takes about 5 minutes.

2.1 Sanitizing raw data (*get_data_from_raw*)

The ***get_data_from_raw*** function is built on two parameters: a vector of node names, and a link to initialize the working directory (where the log-files are).

```
nodes <- c("c0111", "gcn1111", "gcn2222")
main_dir <- "../HPC_preprocessed_logs-files_sec10240606/logs/"
```

To convert the raw data set presented above in a manageable format (data frame), i use the following algorithm:

1. Initialize an empty data frame to gather sanitized data and for each node define the right path and create a list with all files beginning by ”syslog-”.

```
syslog <- data.frame()
dir <- paste(main_dir, nod, sep="")
setwd(dir)
files <- list.files(path=dir, pattern="syslog-", full.names = TRUE)
```

2. For each of these files:

   (a) Only read the lines without decompressing the whole file

```
log_file <- file(data, open="r")  # for each "data" in "files"
file <- readLines(log_file)
```

   (b) Derive data containing only rows with error-messages

```
file <- file[grep("error", file)]
```

(c) Add the year [2] to each line.

```
if(grepl("2023", data)){ file <- paste("2023", file, sep = " ")}
else{file <- paste("2024", file, sep = " ")}
file[grep("Dec", file)] <- gsub("2024", "2023", file[grep("Dec", file)])
# Create two columns two dissociate "date" from sensor "message"
main_file <- str_split_fixed(file, pattern = nod, n = 2)
```

3. Create an additional column with the node name and return the sanitised file.

```
# Iteratively add new gathered data to the initial empty file
  frame <- data.frame(main_file)
  frame$Node <- rep(nod, nrow(frame))
  syslog <- rbind(syslog, frame)
return(syslog)
```

```
> nodes <- c("c0111", "gcn1111", "gcn2222")
> main_dir <- "/Users/engatat/Desktop/test_gpu_cluster/HPC_preprocessed_logs-files_sec10240606/logs/"
> system.time(syslog_full <- get_data_from_raw(nodes, main_dir))
[1] "Start of the collection procedure ..."
[1] "Accessing data in directory /Users/engatat/Desktop/test_gpu_cluster/HPC_preprocessed_logs-files_sec10240606/logs/c0111"
[1] "Accessing data in directory /Users/engatat/Desktop/test_gpu_cluster/HPC_preprocessed_logs-files_sec10240606/logs/gcn1111"
[1] "Accessing data in directory /Users/engatat/Desktop/test_gpu_cluster/HPC_preprocessed_logs-files_sec10240606/logs/gcn2222"
[1] "Collection process completed."
[1] "This process took 5.949 minutes."
      User     System verstrichen
   330.827      7.970    356.972
```

For the whole collection process, the system needed only 8 seconds CPU time and user-tasks about $330.8/60 \approx 5.5$ minutes, for a total of 5.9 minutes.

[2] The year is unfortunately not provided in the data set. So i need to add it manually. For that i consider on one hand the year mentioned in the designation of the file, and on the other hand i always attribute December 2023 and January 2024, because both months can occur in a single file report at the end of the year.

2.2 Extracting different types of failures (*get_errtyp_from_data*)

Now that only failure data (with ”error” mention) is recognisable with dates, sensor mention and the node concerned, the next challenge is to class the failures by different types. Although a large number of lines have been removed up to this point, 926387 lines still remain to go through. I decided to class the failures according to if their source is the software, the hardware or if it is unknown. Therefore i went manually threw the data and collected the most probable and repeated patterns for those classes. The following is a summary of my selection:

Hardware	Software	Unknown
”setup”	”network”	”Link”
”revalidate FID”	”Connection”	”resp_msgs”
”slurm”	”authentication”	”fatal”
”Communication”	”ipmi”	”Unknown”

The function ***get_errtyp_from_data*** is a synthetization of this process. It is composed of three main parts:

1. An error id (”1setup”, ”2revalidate_FID”, ..., ”12Unknown”) and a failure type (”hardware”, ”software”, or ”unknown”) is attributed to each line of filtered data.

```
i = 1
for (err in err_codes) {
  err_id <- grepl(err, filt_log[, 2])
  if (isTRUE(i %in% 1:4 )) {ty <- "hardware"}
  if (isTRUE(i %in% 5:8 )) {ty <- "software"}
  if (isTRUE(i %in% 9:12 )) {ty <- "unknown"}
  filt_log[err_id, "Error_id"] <- paste(i, err, sep="")
  filt_log[err_id, "Fail_type"] <- ty
  # actualize i
  i = i+1
}
```

2. Order data according to the date for later time series analysis and remove the column ”Message”, for it will no more be utilized.

```r
filt_log$Date <- as.ordered(filt_log$Date)
filt_log <- na.omit(filt_log)[, c("Date", "Node", "Error_id", "Fail_type")]
```

3. Let R recognize the Error id-, Failure type-, and Node columns as factors.

```r
filt_log$Error_id <- factor(filt_log$Error_id)
filt_log$Fail_type <- factor(filt_log$Fail_type)
filt_log$Node <- factor(filt_log$Node)
```

The failure attribution process takes approximately 52.43 seconds for the complete data set.

```
> system.time(collect_df <- get_errtyp_from_data(syslog_full))
[1] "Start of the error attribution procedure ..."
[1] "Error attribution process completed."
[1] "This process took 52.43 secondes"
       User      System verstrichen
     44.967       3.701      52.431
> head(collect_df)
                    Date  Node       Error_id Fail_type
1  2023 Dec 12 14:38:20  c0111 4Communication  hardware
6  2023 Dec 12 14:40:28  c0111         1setup  hardware
7  2023 Dec 12 14:41:28  c0111 4Communication  hardware
14 2023 Dec 12 14:45:29  c0111         1setup  hardware
15 2023 Dec 12 14:46:13  c0111 4Communication  hardware
28 2023 Dec 12 14:54:31  c0111         1setup  hardware
```

Once this structural design is operated i have got the data in a well manageable form with already the most important specifications. One may additionally create separated files for different failure types, as it could be convenient for further exploratory data analysis and plots, so one do not make confusion between calculations based on a specific error type containing every nodes and the whole data set.

```r
collect_df <- get_errtyp_from_data(syslog_full)
hard_data <- collect_df[collect_df$Fail_type == "hardware",]
soft_data <- collect_df[collect_df$Fail_type == "software",]
unkn_data <- collect_df[collect_df$Fail_type == "unknown",]
```

3 Extraction, Transformation & Loading

3.1 Statistical Framework

The general failure distribution i.e the probability of the next failure occurring in $[0, t) \subset \mathbb{R}$ is the exponential distribution with a constant failure rate $\lambda \in \mathbb{R}_+$. This means for any variable $X \sim Exp(\lambda)$ we get the cumulative distribution function

$$F(t) = P(X \leq t) = \int_0^t \underbrace{\lambda \, exp\{-\lambda x\}}_{f_\lambda(x)} \, dx = 1 - exp\{-\lambda t\}, \tag{1}$$

where $f_\lambda(x)$ stands for the density function of X and $P(X \geq x + t \mid \ X \geq x) = P(X \geq t)$ expresses memorylessness. In this constellation p- quantiles are defined as $F^{-1}(p, \lambda) = \frac{-ln(1-p)}{\lambda}, \ \ 0 < p < 1$.

For the current context the statistical framework consists processing and estimating downtimes across nodes and failure types. One can compare the next processes to the realisation of a double integral over first the different nodes (because they respect the main [ordered] time-factor), and only then over the different types of failures. If we consider the data as a whole, the double integral can't be switched [3]. To describe the aspects of the probability failure attribution i use the Mean Time Between Failures (MTBF) as main concept for each class i:

$$MTBF_i = \frac{\sum_i (start \; of \; downtime - start \; of \; uptime)}{number \; of \; failures}. \tag{2}$$

So if T is a random variable indicating the time until next failure occurs we get

$$MTBF = E\{T\} = \int_0^\infty \underbrace{t \; f_T(t)}_{Reliability} \, dt \tag{3}$$

and if additionnaly we know $T \sim Exp(\lambda)$, then we can conclude $MTBF = \frac{1}{\lambda}$.

[3] See the theorem of Beppo Levi. As the reliability function will be integrated the "almost sure monotonically growth-" property can't be assured if the nodes are mixed. This can link to negative times between failures (TBF).

3.2 Deriving TBFs and MTBFs (*get_MTBF_from_data*)

In this part i supplement the data obtained in part 2.2 with the time that elapsed between each failure and the last of the same type. This means if an unknown error occurred at $t_1 = 6:00:00\ pm$ and the next of the same type (i.e unknown) happened at $t_2 = 6:00:10\ pm$, then the elapsed time (or easily said the TBF at the 2nd observation) is $\Delta_2 = 10s$ even if ten (10) failures of type "hardware" or "software" or both had occurred in the meantime. According to Equation (2) the sum is built over resp. the nodes to the TBFs per node, then to the MTBF per error types [4]. Of course one could argue that the lost of a component does not necessarily imply the stop of the whole system, but as i said at the beginning, we are not integrating over an unlimited space and the Markov property already is a solution to this argument. For the purpose of automation, i created the ***get_MTBF_from_data*** function, which can be applied in two ways:

- either defining the parameter "param" to be "nodes" (and the function returns a summary table with the numbers of failures, the MTBFs, and the general standard deviations for each node and for each three types of failures),
- or it is set to the parameter "centered_delta". In this case, a new data set is returned with an additional column containing the elapsed time since the error of the same type (TBF), and its centered value by the MTBF of the resp. error class.

The algorithm is as follows:

1. Make a copy of the original data and initialize an empty dataframe for nodes summaries.

```
Data <- data.frame(data)
summari <- data.frame(row.names = c("n", "MTBF","Sigma"))
```

[4] I consider the "start of uptime" in formula (2) as the time of the last error of the same type, because if a failure is not (or at least is not considered to be) repairable by the system, then the whole report process should stop and the report would mention it as the last of that type. It is important to say that all TBF values must necessarily be non negative at the end of this process. It took many time to correct the lines of my codes in an honest manner. It was a good exercise for a better understanding of data transformation, seasonality and different forms of integration.

2. Set the time column to be recognized by the internal time configuration.

```
prev <- Sys.getlocale("LC_TIME")
Sys.setlocale("LC_TIME", "C")
Data$Date <- strptime(Data$Date, "%Y %b %d %H:%M:%S")
Sys.setlocale("LC_TIME", prev)  #"de_DE.UTF-8"
```

3. Iterate over the nodes in two steps.

   (a) Calculate the TBF per node

```
for (j in nodes) {
  nodes_rows <- which(Data$Node == j)
  Dat <- Data[nodes_rows,]
  TBF <- c()
  for (d in 2: nrow(Dat)) {
    Time <- as.numeric(difftime(Dat$Date[d],
                                Dat$Date[d-1],
                                units = "hours"))
    TBF <- cbind(TBF, Time)
    if (is.na(nodes_rows[d]) != T && Time != 0) {
      Data[nodes_rows[d],"Delta"] <- Time  # Adding a TBF column
    }
  }
  TBF <-  c(TBF)
  summari[j] <- c(paste(nrow(Dat)),
                        round(mean(TBF), 3),
                        round(sqrt(var(TBF)), 3))
```

   (b) Assign the MTBF of each class to each error observation.

```
  if(param == "centered_delta"){
    Data[nodes_rows , "MTBF"] <- summari["MTBF", j] }
}
```

4. Make the distinction of outputs between the two possible parameters.

```
if(param == "nodes"){
  return(summari)
}
if(param == "centered_delta"){
  Data$Centered_Delta <- Data$`Delta` - as.numeric(Data$MTBF)
  Data$MTBF <- as.numeric(Data$MTBF)
  return(na.omit(Data))
}
```

Let's now try this function

- with param = "nodes",

```
> # Calculation following error class distinction
> system.time(
+   summari <- list(
+     Hardware = get_MTBF_from_data(hard_data, param = "nodes"),
+     Software = get_MTBF_from_data(soft_data, param = "nodes"),
+     Unknown = get_MTBF_from_data(unkn_data, param = "nodes"))
+ )
   User  System verstrichen
  4.528   0.657       5.758
> summari
$Hardware
       c0111 gcn1111 gcn2222
n       2178    7615    5659
MTBF   1.717   1.296    1.74
Sigma 19.419  13.163  14.759

$Software
       c0111 gcn1111 gcn2222
n        163    3594    1479
MTBF  22.502   2.676   6.597
Sigma 97.113  27.025  25.472

$Unknown
        c0111 gcn1111 gcn2222
n         165     763     519
MTBF   22.124  12.509  14.487
Sigma 124.047 125.712  93.269
```

- and with param = "centered_delta".

```
> # Full data derivation with distinction of error category class
> system.time(
+   data <- rbind(
+     get_MTBF_from_data(hard_data, param = "centered_delta"),
+     get_MTBF_from_data(soft_data, param = "centered_delta"),
+     get_MTBF_from_data(unkn_data, param = "centered_delta"))
+ )
   User  System verstrichen
  3.889   0.605       4.816
> head(data)
                  Date  Node       Error_id Fail_type  Delta (Δ)  MTBF Centered_Delta
6  2023-12-12 14:40:28 c0111         1setup  hardware 0.03555556 1.717      -1.681444
7  2023-12-12 14:41:28 c0111 4Communication  hardware 0.01666667 1.717      -1.700333
14 2023-12-12 14:45:29 c0111         1setup  hardware 0.06694444 1.717      -1.650056
15 2023-12-12 14:46:13 c0111 4Communication  hardware 0.01222222 1.717      -1.704778
28 2023-12-12 14:54:31 c0111         1setup  hardware 0.13833333 1.717      -1.578667
29 2023-12-12 14:55:39 c0111 4Communication  hardware 0.01888889 1.717      -1.698111
```

The execution of this procedure amounts $5.8\ s$ for the whole data set with the "nodes" parameter. param="centered_delta" needs even less than that i.e $\approx 4.8\ s$.

Note: The function ***get_MTBF_from_data*** is implemented such that it is even possible to apply it on non disjoint nodes data. All values will still be non-negative, but then this would be equivalent to a marginal approach, because it is then assumed that all nodes have the same error distribution. My approach is rather conditional, as i put myself before the next failure occurrence and challenge it's probability by historical clustered data. In the following i show how this is important.

3.3 Exploratory Data Analysis

From now on, we only assume non-negative values. A negative value would mean a failure in one of the precedent implementation processes. I test this with the following command:

```
1 which(data$`Delta`<=0)
```

The response should be "**integer(0)**", meaning that no such line was founded. Let us now gain further insights from the pre-processed and transformed data.

A quick overview can be attained by looking at the failure behavior of the nodes with different types of failure. Figure 2 shows that all three nodes have approximately the same number of errors (observations). One can get an idea of how the frequencies are distributed over the groups if rather pre-defined ID-failures from the table in part 2.2 is considered. The first four IDs are linked to hardware failures, the next four to software- and the last four to unknown failures. The first, third, fourth, sixth, and seventh types of error have a relatively bigger weight, specifying rather non-uniform cumulative distribution over failure classes. In general one can just say that there are more hardware than software failures on the cluster, and more software failures than unknown. This does not mean that the MTBF follows the same order. We shall see in the following that it does not necessarily holds. To be clear about this fact and to have a clear understanding of the general failure distribution, i will represent frequencies, MTBFs and standard deviations later.

Before switching to MTBF and standard deviation per group, let's compare the failure distribution on a lower with on a larger time scale. Figure 3 shows the results over a time scale of around 5 months i.e in between $[0, 3300\ hrs)$. There i

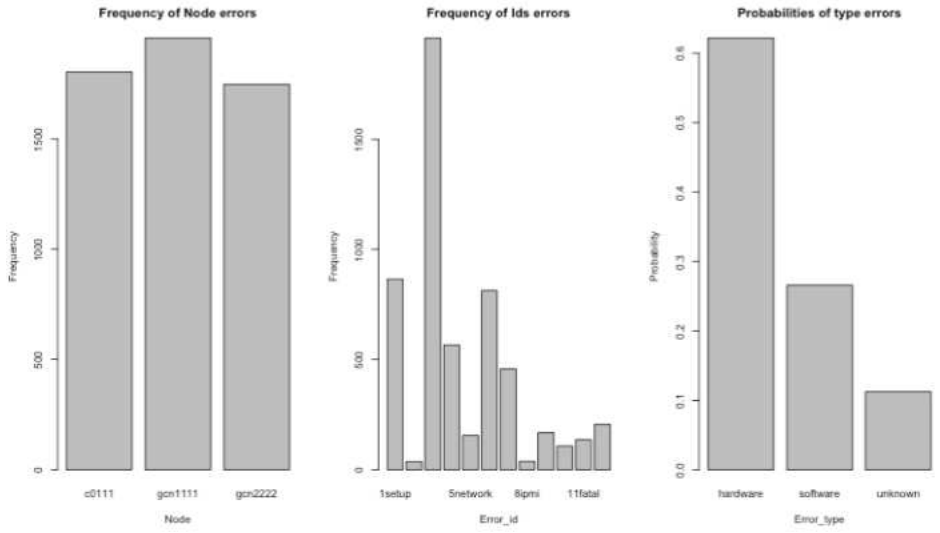


Figure 2: Failure Frequencies for nodes, id errors and probabilities of the error types.

directly observed that the hardware failures are comparatively more concentrated around 0. Unknown failures are rather more sporadic and therefore will probably give the largest within group variance. In distribution all types seems to behave the same, but it also seems obvious, that software are somehow more correlated to hardware than to unknown errors. A reason for than could be the fact that a great period (before December 2023) is missing in the ”c0111” node repport (See Figure 6 from April 2023 to December 2023). I predict therefore that the standard deviation will approximately follow the failure distribution if more error classes were considered. On a lower time scale i.e $[0, 48\ hrs)$ (See Figure 4), it is obvious that software and unknown errors have the biggest variances as well.

To understand what the outages look like in a smaller time interval, e.g. between one and two days, the following box-plot is constructed. From there already, it becomes clear that the time between failures is much lower on the ”c0111” node, whereas on the other two nodes ”gcn1111” and ”gcn2222” failures seems to degenerate from a particular break point. More observations can be made, but to be precise about such kind of estimation, it is suggested to perform a precise statistical model based on proper assumptions on the distribution of the error terms.

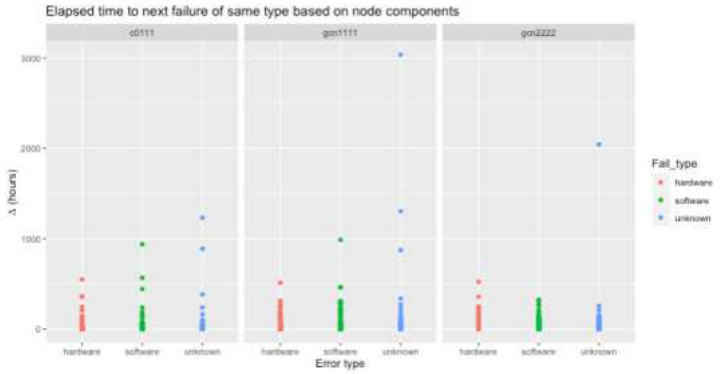


Figure 3: Larger scale visualization of Times Between Failures.

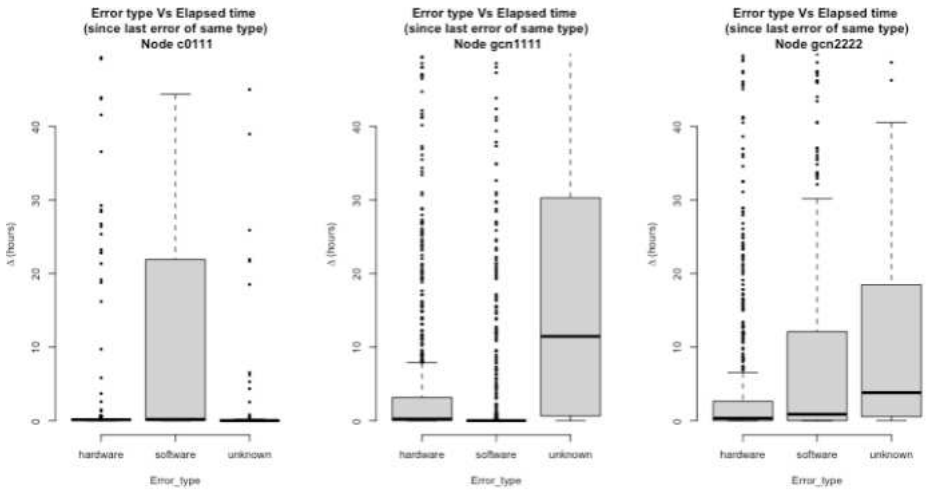


Figure 4: Lower scale Boxplot visualization of Times Between Failures.

Figure (5) is to get a fast insight into group correlations based on determinant statistical measures. To derive asymptotic inference for various groups one could take advantage of the central limit theorem. This is, if $Y_1, ..., Y_n$ are i.i.d random variables with $E(Y_i) = \mu$ and $Var(Y_i) = \sigma^2$, then $\sqrt{n}\frac{\overline{Y}-\mu}{\sigma} \overset{a}{\sim} \mathcal{N}(0,1)$, $\overline{Y} = \frac{1}{n}\sum Y_i \overset{a}{\sim} \mathcal{N}(\mu, \frac{\sigma^2}{n})$ and $\sum Y_i \overset{a}{\sim} \mathcal{N}(n\mu, n\sigma^2)$.

Figure (6) summarizes all the above details. There I perform gamma density approximations with B-splines over the available nodes and failure modes. These estimations are already a great start for further predictions but still i recommend an MCMC implementation at this point to confirm/emphasize expectations/results.

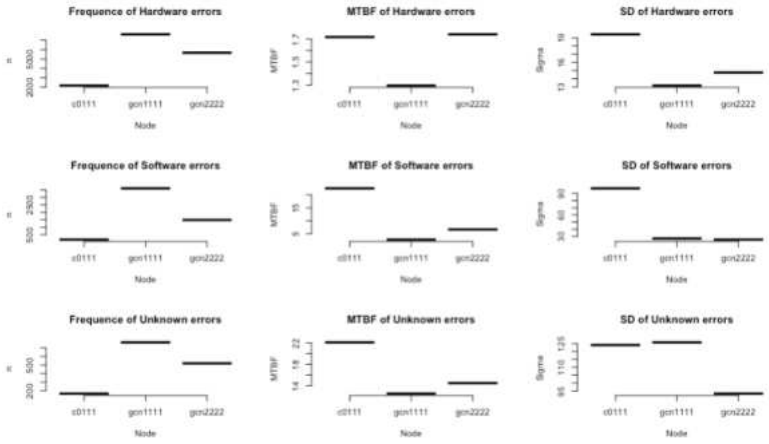


Figure 5: Results of the data transformation procedure.

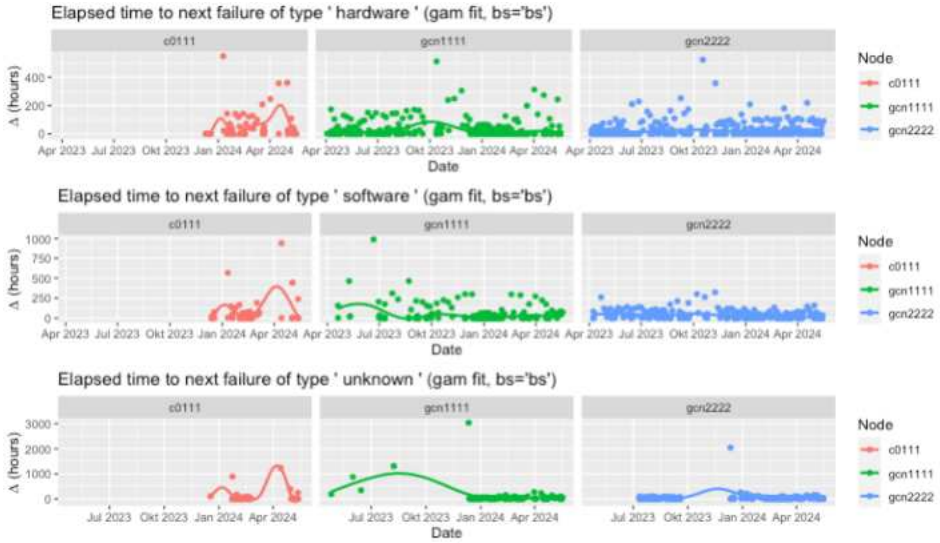


Figure 6: Failure density estimations with a gamma regression including B-splines.

4 Models and Applications

4.1 Simple MCMC Implementation

A Markov chain (MC) or Markov process is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous one. It coincides exactly with the memorylessness property from formula (1) and the MTBF definition from formula (2). Given the probability distributions expressed in Figures 2 and 5, an MCMC (Markov Chain Monte Carlo Simulation) can be used to draw samples from the true probability distribution. For a statistician this is that "the MC equilibrium distribution matches the target distribution" [5]. The more steps are included, the more closely the distribution of the sample matches the actual desired distribution.

```
library(rstanarm)
library(bayesplot)
y <- data$`Delta`
v <- as.factor(data$Fail_type)
w <- as.factor(data$Node)
glm <- glm(y ~ v+w,          #### Simple GLM
           data = data,
           family= gaussian(link="log")))
bglm <- stan_glm(y ~ v+w,    #### MCMC (GLM BAYESIAN ESTIMATION)
            data = data,
            family = Gamma(link="log"),
            prior_intercept = normal(
                mean(Re_sults$MTBF, na.rm = T),
                mean(Re_sults$Sigma, na.rm = T)^2/nrow(Re_sults)),
            chains = 4, iter = 300*2, seed = 1915,
            prior_PD = F)
```

[5] The MCMC process represent a big asset to confirm theoretical assumptions on the error terms and the failure distribution as well. The target here is so to say the "difference" between theoretical and empirical measures. Predictions are then based on an $\alpha\%$ Confidence interval.

To implement MCMC in R i use the ”stan_glm” function from the **rstanarm** R-package with the gamma distribution family with log link, 4 chains, 600 iterations, i set the prior intercept to the theoretical distribution of the mean from the central limit theorem (i.e $\overline{Y} = \frac{1}{n}\sum Y_i \overset{a}{\sim} \mathcal{N}(\mu, \frac{\sigma^2}{n})$) and compare the results with those of a generalized linear regression with Gaussian (normal) priors. From the **bayesplot** package i am able to visualize trace-, densities and area plots with resp. the functions ”mcmc_trace” (Figure 8), ”mcmc_dens_overlay” (Figure 9) and ”mcmc_areas” (Figure 10). To make predictions for particular types of failures and/or nodes i highly recommend the ”pred_bglm” function from the same library.

```
> cbind(bglm=bglm$coefficients, glm= glm$coefficients)
                 bglm       glm
(Intercept) 1.2194606 1.7872635
vsoftware   0.8183877 0.5707205
vunknown    1.6043899 1.5836663
wgcn1111    0.8928766 0.4758630
wgcn2222    0.9864893 0.0288229
```

Figure 7: Comparison between simple glm and mcmc results.

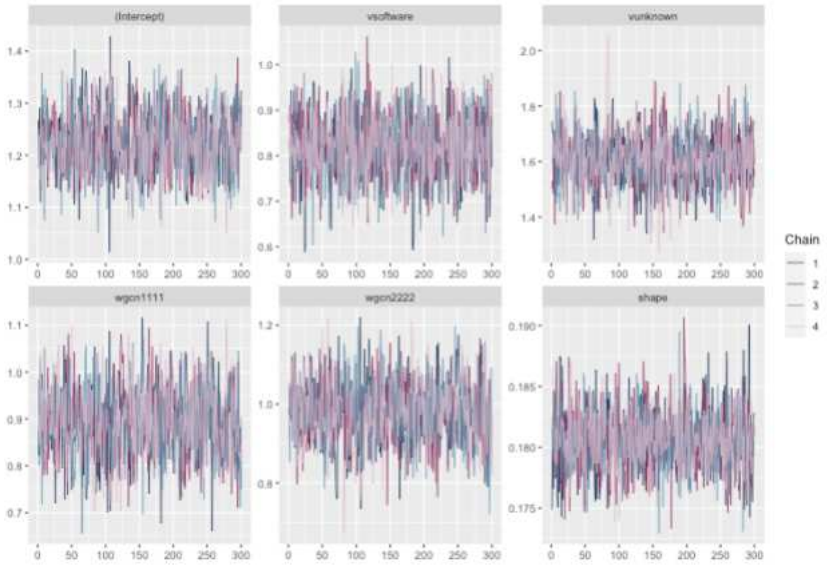


Figure 8: Trace plots of MCMC chains.

The intercept can be viewed as a model approximation for the response variable when each covariate is set to zero. In distribution, this can be compared to the estimation of the MTBF on the cluster if one would only concsider hardware reports from the ”c0111” node as the number of iteration grows. It is possible to configure the MCMC to calculate without an intercept and the hardware types of failures would also be aggregated. In Figure 8, the estimated means and standard deviation are readable on the y-axis and the number of iterations on the x-axis.

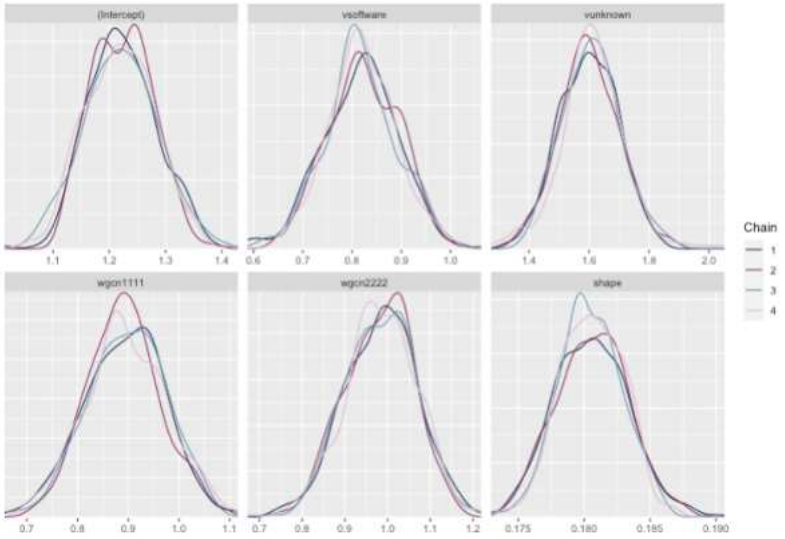


Figure 9: Densities of MCMC chains.

Figure 9 represent the transformation of the same information in densities. It is easy to see from their shape that the choice of Gaussian priors was a good one. Since i made the Gaussian prior assumption for the the error terms and the Gamma for the regression family, it is recommended first to test the prior assumption(s) before moving forward to the derivation of confidence/prediction intervals. I do it with the ”pp_check” function of the same ”bayesplot” package. The top left histogram represent the true distribution over the number of iterations. Interval estimates are made in Figure 11 so one gets a clear vision of the difference in generated MCMC mean times (MTBFs) across nodes and failure types.

```
pp_check(bglm, plotfun = "hist", nreps = 5) +  xlab("Delta")
```

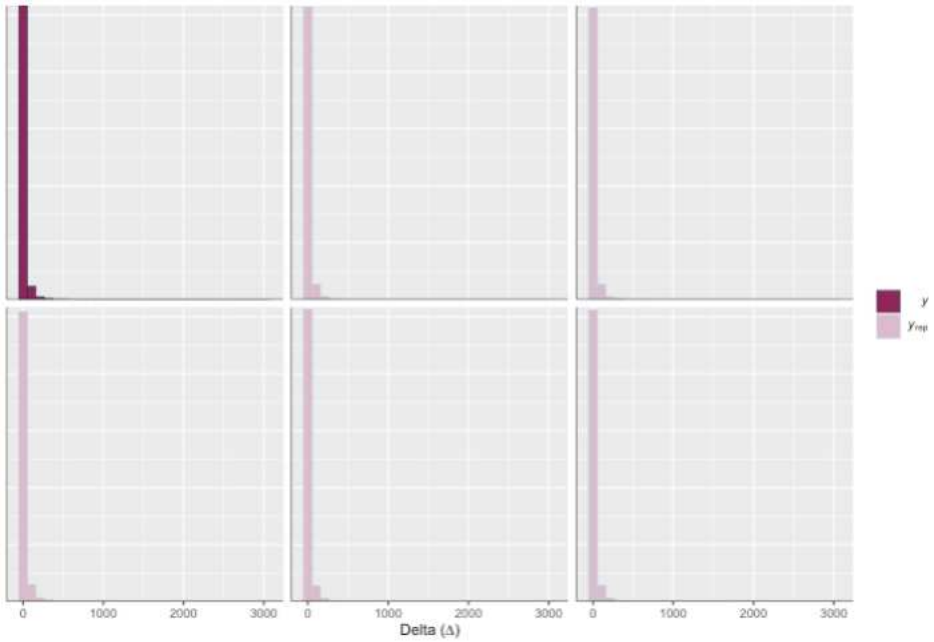


Figure 10: Prior predictive distribution check.

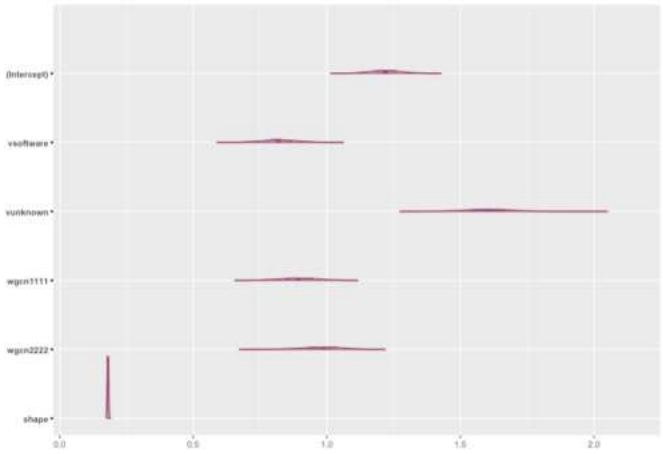


Figure 11: Interval estimates from MCMC draws.

4.2 Mixed Modelling

Mixed models are typically employed for the analysis of grouped data. Call them mixed models or models with random effects, still you will consider output variations for random slopes, random intercepts or both. In this part i consider two different sources of variation in the data: on one hand the variation within groups and on the other the variation between groups. I assume variations from the true elapsed times to be normally distributed as for an auto-regressive process of order 1 [6]. Then, i take the latest time series approach and base on it to compare a random intercept to a random slope model with help of the "lme" function of the **nlme** package [7]. I compare the results to a simple linear approximation and to a generalized additive model with B-splines extended by confidence intervals (See figure 13). I explain the conditional and the marginal perspectives in this configuration and briefly talk about the ICC (Intraclasscorrelation coefficient) as measure for the proportion of within-group variance relative to the total variance.

```
x <- as.Date(data$Date)
lm <- lm(y~x, data=data)
ri <- lme(y~x,random = ~1| w, data=data)
rs <- lme(y~0+x, data=data, random =  ~1+x| w,
          control = lmeControl(opt="optim"))
data$lm <- fitted(lm)
data$ri <- fitted(ri)
data$rs <- fitted(rs)
```

The results for both normal elapsed times between failures and centered values around each their MTBF as shown in resp. Figure 12 and 13. Depending on the

[6] In statistics, an auto-regressive process of order 1 [AR(1)] is specified with $\epsilon_{ij} = \alpha\epsilon_{i,j-1} + u_{ij}$ i.e the variation obtained for two observations are linearly correlated by the factor of a value $-1 < \alpha < 1$, and the errors u_{ij} of such an approximation is normally distributed with mean 0 and with the within group variance.

[7] Here i consider only the nodes as groups. It it possible to replicate my example by interchanging the nodes with the types of failures. Comparing results with the MCMC presented above, on will get a great understanding of the failure and data behaviour.

goal, one can intuitively decide which model approach adapts/fits the best [8].

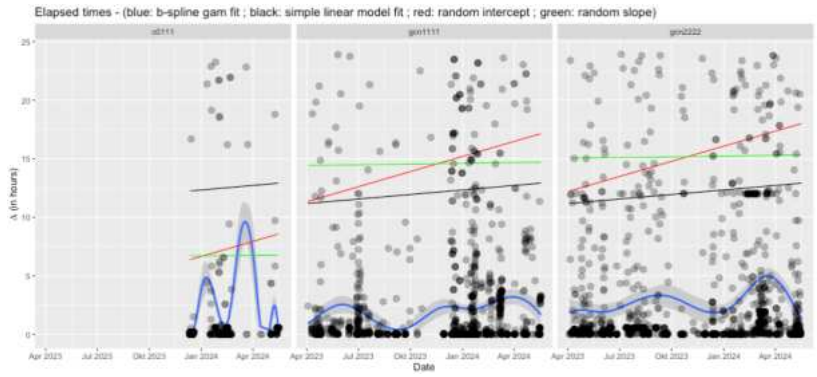


Figure 12: Several fits on TBFs.

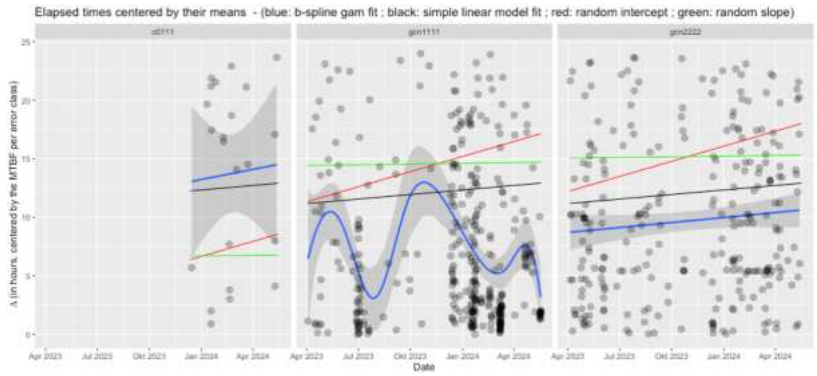


Figure 13: Several fits on MTBFs, i.e conditional perspective on both nodes and error types.

Now lets numerically compare results from the random intercept with those of the random slope model, and try a conditional and/or marginal approach on the data.

[8] The ICC (Intraclass Correlation Coefficient) stands for how the variance variates across multiples groups. It is not derived in the same way for random intercepts as for random slopes. Therefore its interpretation should be done very carefully, as the total variance are stored differently on each co-variate.

```
> (s_ri <-summary(ri))
Linear mixed-effects model fit by REML
  Data: data
       AIC     BIC    logLik
  62479.94 62506.4 -31235.97

Random effects:
 Formula: ~1 | w
        (Intercept) Residual
StdDev:    5.441411 70.12339

Fixed effects:  y ~ x
                Value Std.Error   DF   t-value p-value
(Intercept) -264.94371 184.78001 5505 -1.433833  0.1517
x              0.01407   0.00938 5505  1.499812  0.1337
 Correlation:
  (Intr)
x -1

Standardized Within-Group Residuals:
        Min          Q1         Med          Q3         Max
-0.25653222 -0.20644906 -0.13879436 -0.08954275 43.08586088

Number of Observations: 5509
```

```
> (s_rs <- summary(rs))
Linear mixed-effects model fit by REML
  Data: data
       AIC     BIC    logLik
  62496.28 62529.35 -31243.14

Random effects:
 Formula: ~1 + x | w
 Structure: General positive-definite, Log-Cholesky parametrization
            StdDev       Corr
(Intercept)  4.448400283 (Intr)
x            0.000181153 -0.226
Residual    70.131939878

Fixed effects:  y ~ 0 + x
        Value    Std.Error   DF  t-value p-value
x 0.0006195885 0.0001552025 5506 3.992129   1e-04

Standardized Within-Group Residuals:
        Min          Q1         Med          Q3         Max
-0.21837387 -0.20811822 -0.14280564 -0.09316465 43.08521079

Number of Observations: 5509
Number of Groups: 3
```

Here one sees the difference in random slopes versus random intercepts estimates. While random intercepts have a intragroup variance (from within) of $5.44^2 = 29.59$, random slopes propose a within group variance of $4.45^2 = 19.8$. For the between group variance one gets approximately the same value $70.12^2 \approx 4916.8$ for both random intercepts and slopes models. Still it is lower than the number of observations. The AIC of the random slope model (62496.28) is greater than the intercept's one (62479.94), indicating a preference for the model with random axis intercepts [9]. Personally i think the generalized additive model fits the data best, in particular when it comes to quick decision-making. For the random intercept model, the ICC is defined as $\frac{29.59}{4916.8+29.59} \approx 0.006$. When the ICC tends to 0 then a simpler model is more suitable (e.g in this case the simple linear regression or the random slope model).

In this work i do not cover the ICC for random slope models, but at least from the above derived measures, it is already possible to decide which one is better in an hierarchical manner, and in consideration of the job-partition one would like to estimate.

[9] The AIC (Akaike Information Criterion) is a statistical measure that penalizes the inclusion of additional variables to a model. It adds a penalty that increases the error when including additional terms. The lower the AIC, the better the model.

5 Conclusions

The main idea of mixed modelling is to consider either random intercept or slope models to derive variances within and between groups. Estimates for conditional and marginal perspectives will diverge most of the time, but it is important to make this distinction to derive appropriate interpretations. It is not excluded that a more specified analysis will be carried out and further details discussed, such as how the implemented functions can be made available in the form of an online package or software for the benefit of HPC users. For the moment i was able to gain a deep insight in the understanding and modelling of various failure distributions and types, especially in the framework of HPC clusters. I showed how data can be extracted in a statistical framework based on mean times between failures and explained a possible integration process. I also was able to perform exploratory data analysis based on self pre-processed data with approximately reliable results. I described how the MCMC algorithm works. It shall be, may be an interesting experiment to see how fast my algorithm would perform directly on a HPC cluster with more error types and boosted by much more nodes, since the here described implementation was completely static.

Interesting bibliography on modeling failure statistics:

- *GRAAFE: Graph Anomaly Anticipation Framework for Exascale HPC systems* (M. Molan, M. S. Ardebili et al, 01.02.2024).
- *Reliability and Residual Life of Cold Standby Systems* (L. Liu, Xiaochuan Ai, and Jun Wu, 18 April 2024).
- *Comparative analysis of soft-error sensitivity in LU decomposition algorithms on diverse GPUs* (G. Leon, J. M. Badia, et al, 22.02.2024).
- *Failure probability estimation with failure samples: An extension of the two-stage Markov chain Monte Carlo simulation* (S. Xiao, W. Nowak, 15 April 2024).
- *Regression - Modelle, Methoden und Anwendungen* (L. Fahrmeir, T. Kneib & S. Lang, 2022).
- *Multilevel Modeling Using R* (W. H. Finch, J. E. Bolin, K. Kelley, 2007).