Master thesis in Industrial and Management Engineering

Change detection and identification in a simple serial line

A manufacturing application of Process Mining

By

Edoardo Chiò

******

Supervisor:

Prof. A.Alfieri

Politecnico di Torino

2020

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Edoardo Chiò
2020

Acknowledgements

And I would like to acknowledge ...

Abstract

In production systems, digital twins must be always aligned with the real system to guarantee an effective decision making process in a continuously changing environment. To allow the alignment, digital models can be updated with process mining techniques through data collected by sensors. This paper addresses the issue of detecting changes in the production system by analyzing data collected from sensors. Using raw collected data, a procedure is proposed to compute and plot relevant system measures that could help change identification. Simulation is used to test the effectiveness of the procedure in a realistic medium size production line.

Keywords: production systems; process mining; change identification; Industry 4.0; digital twin

Contents

List of Figures

List of Tables

Nomenclature

Roman Symbols

F	complex function

Greek Symbols

γ	a simply closed curve on a complex plane
$\imath$	unit imaginary number $\sqrt{-1}$
π	$\simeq 3.14\ldots$

Superscripts

j	superscript index

Subscripts

0	subscript index

Other Symbols

$\oint_\gamma$	integration around a curve γ

Acronyms / Abbreviations

CIF	Cauchy's Integral Formula

Chapter 1

Introduction

In the last three centuries humankind lived three different industrial revolutions that deeply modified every aspect of societies, not only changing production and economic models, but also generating new cultural movements and wide political unrests across the whole world. The First Industrial Revolution began at the end of the 18th century in Great Britain. The design of efficient steam engines and the use of coke as fuel allowed to highly increase production, especially in iron making and textile industries, and to reduce transportation cost and time. After a slowdown in important innovations in the middle of 19th century, in the last decades of the century many new inventions (e.g. electrical power, petroleum refining, rubber vulcanization) produced a continuous improvement in manufacturing industry. This period is known as Second Industrial Revolution and culminated with Henry Ford's product standardization and mass production, inspired by Taylor's "The Principles of Scientific Management". Ford's model (known as Fordism) brought to a massive production and consumption increase, generating both huge wealth and widespread social tensions (?).

At the end of the 1940s the transistor was invented in AT&T Bell Laboratories. This was the first step to the design and production of microprocessors which are the basis of the Third Industrial Revolution. In particular, in the last decades of 20th century computers led to process automation and system control and monitoring, causing an increase in productivity, efficiency and quality.

Processors capabilities development and diffusion in every field of human activity has gone along with data storage capacity increase, which in recent years led to the problem of information extraction from huge, complex, and scattered databases.

Manufacturing has been deeply influenced by the recent advancements in digital technologies. Innovative production and process control techniques started to spread in the last decade; the most remarkable ones are cyber-physical systems (CPSs), Internet of Things (IoT), digital twin,

big data analytics, and cloud computing [citare Intelligent Manufacturing in the Context of Industry 4.0: A Review]. The challenge of implementing the digitalization in manufacturing led to define a new high-tech strategy by the German government in 2011, soon taken as example by the rest of the world. In the same year, the term "Industrie 4.0" (i.e. Industry 4.0) was introduced [citare J. Jasperneite, Was hinter Begriffen wie Industrie 4.0 steckt, Internet Und Autom. 12 (2012) 12], implying that these technological developments are considered so significant that a Fourth Industrial Revolution is starting.

1.1 Industry 4.0 components

To understand the complex landscape of Industry 4.0, definitions of the main facets of this new manufacturing model are presented and a relationship between them is proposed.

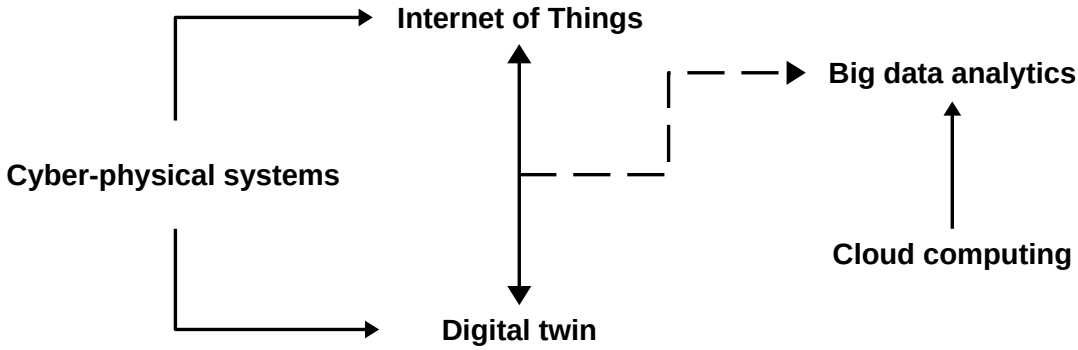


Fig. 1.1 Industry 4.0 scheme

Cyber-physical system (CPS) A CPS is defined by [citare A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems] as "transformative technologies for managing interconnected systems between its physical assets and computational capabilities". In other words, CPS refers to a system where physical objects and software elements are linked to interact and exchange information. In [citare A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems] two main functional components of a CPS are identified, which are (a) an advanced connectivity able to manage data acquisition and transfer in real-time, and (b) a cyber-space where information is extracted and consequent decisions are taken. Then in the same paper a 5-level ("5C") architecture is proposed as guideline for implementation of CPS in a manufacturing application.

Internet of Things (IoT) The term IoT refers to a production or service model where various sensorized objects (i.e. "Things") are connected, creating a network through which they collect and share data in real-time. The implementation of an IoT system introduces many challenges. A reliable and affordable automatic identification technology is required to made the objects "smart"; RFID technology is a viable answer, and it's already actively used for identifying various objects in warehouses, production shop floors, logistics companies, distribution centers, retailers, and disposal/recycle stages. Also, a secure, high speed, and high bandwidth wireless communication standard has to be adopted to manage the huge amount of data generated by the sensors embedded with the objects; 5G technology is a possible, yet controversial, candidate to address this problem.

Digital twin (DT) The DT is "the virtual and computerized counterpart of a physical system that can be used to simulate it (i.e. the physical system) for various purposes, exploiting a real-time synchronization of the sensed data coming from the field" [citare A Review of the Roles of Digital Twin in CPS-based Production Systems]. The deep difference with a simple simulation model is that a DT is not a static but a dynamic representation (at a desired level of detail) of the real system, able to imitate its changes and influence it, while running in parallel. More precisely it can be said that the DT is the software side in a CPS.

Big data analytics Big Data, as written in the HACE theorem [citare Data mining with big data], "starts with large-volume, heterogeneous, autonomous sources with distributed and decentralized control, and seeks to explore complex and evolving relationships among data". These characteristics are typical of any data output in CPS, where different sensors spread across the production lines gather and send data to a central system. The amount of collected data requires the use of modern machine learning techniques to extract useful information.

Cloud computing Cloud computing is a general term that indicates the offer by Internet-related companies (like Google, Microsoft, or Amazon) of computational and storage services through scalable resources over the web. The scalability of resources allows organizations to reduce initial investments and to adjust the business digital capabilities as needs arise. The most significant concerns about cloud computing, apart from technological challenges such as load balancing, scalability and availability, and compatibility among different clouds, are related to privacy issues and security.

1.2 Industry 4.0 tools

Data mining and process mining In broad terms, the implementation of Industry 4.0 paradigm has two sides: on the hardware side, the embedding of sensors in the production lines and the adoption of a high bandwidth wireless communication mean. On the software side, the use of analysis tools which can extract information from the huge amount of data gathered by the sensors and the engineering of a system capable of exploit the information to build a digital copy of the real line. In the latter part, as mentioned before, Machine Learning (ML) algorithms and Data Mining (DM) come into play. Machine learning is a relatively recent mathematics and computer science domain [citare Machine learning in manufacturing: Advantages, challenges, and applications] The activity of obtain information from big data volumes is generally called knowledge discovery in databases (KDD), defined in [citare Data Mining in Manufacturing: A Review Based on the Kind of Knowledge] as “the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data”. KDD includes many different steps, such as data preparation, data selection, data cleaning, incorporation of prior knowledge, and result interpretation. The core of this process is Data Mining (DM), which is the application of specific algorithms for identifying patterns in data to uncover hidden relationships and predict outcomes.

Chapter 2

Scope of the work

A recently published article by [citare Manufacturing systems mining: Generation of real-time discrete event simulation models] proved that a static process model (represented using Petri Nets, a process modelling language), starting from an event log, can be built using PM algorithms properly adapted to a manufacturing industry application. Thereafter, a static DT can be created using the resulting process model. However, as said in chapter **??**, a Digital Twin aims to be a dynamic copy of a real manufacturing system. It means that, ideally, the DT is changed as soon as in the real system a variation occurs, in order to keep it continuously aligned.
In this work the attention is directed on laying the foundation for an always up-to-date DT, making it an auto-updating copy of the system given an initial process model.
The starting point is to define the three main steps that compose the model update process when a variation in the real system occurs.

- **Change detection**: The production line is monitored to verify if a variation is occurring during the process. It is performed in real-time, computing and analyzing indicators extracted from sensor output. The continuous inspection of these indicators can be accomplished using statistical control process (SPC) techniques (e.g. Shewart Charts, Cusum Charts, EWMA Charts, Changepoint Detection), which indicate if the process behavior in a certain instant significantly differs from previous observations. So, if in the real system a variation takes place, it is possible to notice and report the change occurrence with a predetermined accuracy level, that depends on the sensor reliability and on the chosen significance level used in statistical tests.

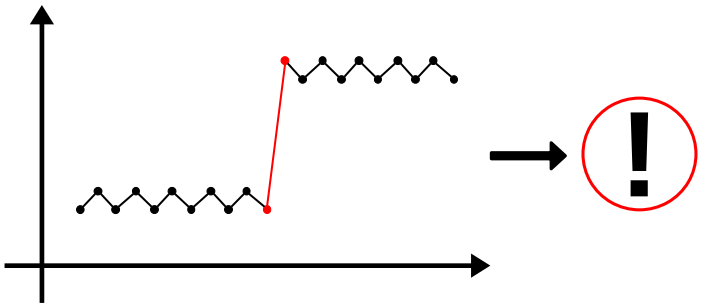


Fig. 2.1 Model update process Detection

- **Change identification**: A detected change in one or more indicators is classified to determine what kind of variation occurred in the underlying process. To do so, a "change map" is consulted, in order to match the indicator detected change with a real system change, and then choose the correct update to the process model.

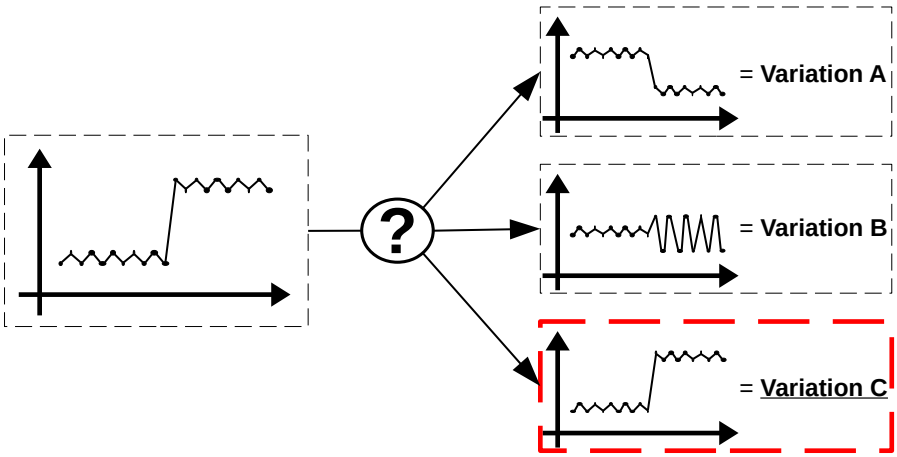


Fig. 2.2 Model update process Identification

- **Model modification**: The process model is modified accordingly to the identified variation. A radical way to update the model is to build it from scratch, using PM algorithms, every time a variation occurs; however, in case of limited or local changes, it is not recommended to use this procedure, since the computational time and effort could be excessive. For this reason, a better approach could be designing algorithms more specific to the occurring variation type and position, aiming to a focused rather than general change in the model.

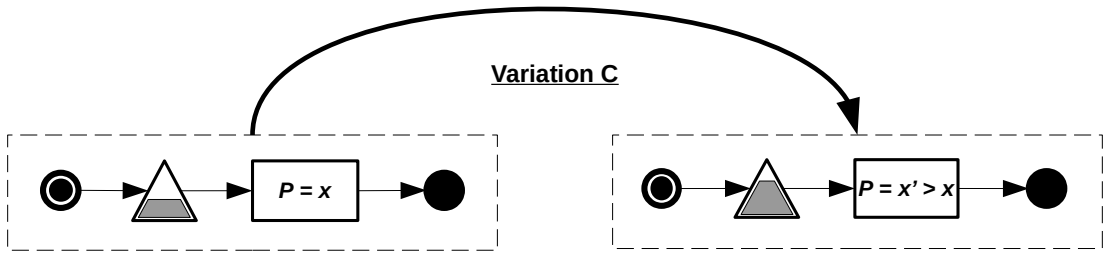


Fig. 2.3 Model update process Modification

In this chapter, firstly the goal of the work is stated. Then, event logs are described, that are the data source from which indicators are computed. In the remaining sections, the scope limits are defined, concerning the type of systems, sensors and variations that are considered in this thesis.

2.1 Aim of the work

The thesis focuses on the second step of the model update process and, partially, on the first step. Concerning the change detection, this thesis aims to answer whether or not a certain type of variation in the system is noticeable observing the monitored process indicators. Detection methods usage and even which one to apply fall outside the scope of the work. In other words, the focus is on if a change can be detected, not how to detect it.
At the same time, the study addresses the change identification topic, checking if it is possible to recognize the type of variation through the analysis of indicator behaviors. The main goal is to build a variation map that can be used as a guide to get insights on the process evolution.
This thesis is devised as part of the PM framework. In this sense, the observed indicators are extracted only from event logs generated by line sensors. The purpose of this choice is to base the model update on the same data used to build the model in the first place, without requiring the addition of different sensors and the processing of other data structures.

2.2 Event logs

An event log is a data table referring to a specific process, where each row records a single event occurred during the process. Events are sequentially registered such that, at least, each one is associated to the case (i.e., a process instance, a job) object of the event, and with the activity (i.e., a well-defined step in the process, e.g. entering in a buffer, starting the production) performed

on the case. [citare R.P. Jagadeesh Chandra Bose, R. Mans, and W.M.P. van der Aalst. Wanna Improve Process Mining Results? It's High Time We Consider Data Quality Issues Seriously. In B. Hammer, Z.H. Zhou, L. Wang, and N. Chawla, editors, IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2013), pages 127–134, Singapore, 2013. IEEE].
Thus, if the event order is preserved, a minimal event log (called simple event log) is composed of only two fields, containing the following information

- **ID of the case**: code that uniquely identifies a certain case flowing in the process.
- **ID of the activity**: code that uniquely identifies a certain activity performed on a case

With these three information (including the event order) a trace can be extracted for each case, which is the ordered sequence of activities executed on a case. So, a simple event log can be reorganized as a multi-set of traces over a set of activities.
Richer event logs contain more attributes, such as

- **Timestamp**: instant when event verified
- **ID of the resource**: code that identifies which resource (e.g. an operator that performs different activities, a machine in a multi-machine stage) is utilized
- **Cost**: cost related to the event
- **Case type**: category to which the case belongs, useful in a multi-product system

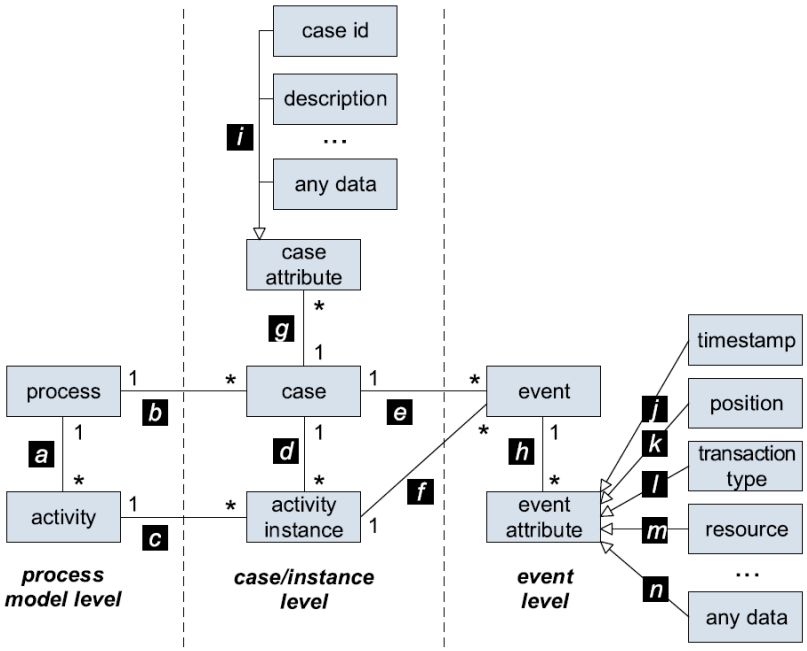


Fig. 2.4 Event log class diagram [citare libro]

2.2.1 Event logs in real applications: event streams

In manufacturing, event logs are the output produced by sensors positioned along the production line. The sensor locations are determined a priori, and the sensors should guarantee a good level of reliability to minimize errors in the event log.

It is to be noted that, in real applications, event logs often do not present as complete datasets, instead they are generated continuously appending new events recorded in real-time by line sensors. In other words, an event stream is collected in an always-extending event log and there is no point in time where the dataset is finished, including the entirety of occurred events. A real-time event gathering causes challenges that basic Process Mining algorithms are not able to deal with: datasets must be finite to be studied, and arbitrarily cutting the event flow can produce incomplete traces and not allow to see possible variations in the process. The Process Mining branch that aims to address this problem is called Streaming Process Mining; different solutions have been proposed, among them the most efficient are the usage of smarter forms of sampling, and summarizing the data in event frequency tables.

This thesis only considers event logs consisting of complete datasets, while the event stream situation is not explicitly taken into account. This simplification causes a partial, but not total,

generality loss with respect to the event flow case. Indeed, this dissertation has two objectives, that are to understand if it is possible to detect and identify some specific types of variation. Concerning the change detection, since this problem is addressed studying the event log with an event-by-event (or window-by-window) approach, that is comparing every new event with the previous ones, the dataset completeness is not necessary and the same approach could be also applied in an event stream context without any adaptation. Instead, regarding the change identification, the event log completeness is much more relevant. To understand what kind of variation is occurring, a simple comparison of consecutive events is insufficient, and the complete event log case and the event stream case lead to different approaches. In the first case (if the event log includes an enough long collection of after-change events), the dataset serves as a snap-shot of the process, making easy to recognize steady state periods before and after the variation and allowing to directly compare them to get insight about the process evolution. Instead, the second case, which necessarily works collecting events (or event windows) one at a time, must also deal with the problem of identifying when the process reaches again the steady state after the variation.
Therefore, while reasonings about the change detection can also be straight applied to event stream situations, even if the study only considers complete event logs, the change identification presents additional challenges that are not addressed in this thesis.

2.3 Considered systems

2.3.1 Simple serial lines (SSLs)

In this thesis the considered systems are limited to simple serial lines (SSLs), also called basic straight lines, with one resource for each of stage. SSLs are described in [citare Serial production line performance under random variation: Dealing with the 'Law of Variability', Romero-Silva R], having the following characteristics.

- **Production in discrete parts**: Discrete manufacturing involves finished products consisting of distinct items (or batches of items) that can be counted, touched, or seen. Production is guided by a bill of materials (BOM) and follows a route, such as an assembly line. Examples of discrete manufacturing products are automobiles, furniture, and smartphones. In theory, a discrete product can be broken down at the end of its lifecycle so its basic components can be recycled. Discrete manufacturing contrasts with process manufacturing, where the product is created by using a formula or recipe to refine raw ingredients and the

final product cannot be broken down to its basic components. Examples of goods produced by process manufacturing include pharmaceuticals, food and beverages, refined oil and paints. [citare https://searcherp.techtarget.com/definition/discrete-manufacturing]

- **Single type of product**
- **Production and transfer batches composed of one unit**
- **FIFO (First In First Out) service policy**: This service policy (i.e. rule which defines the resource feeding order from the preceding buffer) prescribes that in each stage the first job to enter in the buffer is the first to be processed by the resource(s). The exact opposite of FIFO is LIFO (Last In First Out), which prescribes that the last job to enter in the buffer is the first to be processed by the resource. Since FIFO service policy is implemented in all stages and each stage is composed by only one resource, the job arrival order, the job process order in every stage and the job disposal order is always the same. In other words, jobs cannot swap positions after the arrival in the line. Therefore, each job is uniquely identified by its position.
- **Not synchronized operations**: Each station's operation is independent from the others. The only information that a resource has access to are the fill state of the preceding buffer (i.e. the buffer in the same stage where the resource is located) and the following buffer (i.e. the buffer in the next stage). If the preceding buffer is empty starvation occurs, that is the production stop caused by the absence of jobs to process. On the other side, if the following buffer is full, blocking occurs, that is the production stop caused by the lack of space to unload the job processed by the resource. Lack of synchronization in operations also prevents setups planning to minimize completion time.

2.3.2 Production line characteristics assumptions

To focus on specific types of manufacturing system structures, further assumptions have been done

- **Production line is initially stable**: Stable lines are production lines where the job average arrival rate is lower than the system average service rate. In other words, mean interarrival time is higher than mean processing time of the bottleneck. Because of this, true bottleneck of the system is the arrival process. This assumption is made to consider more realistic production situations: an initially unstable system is clearly out-of-control since the

beginning of monitoring, so it is not worth to be studied using methods that aim to find subtle processing problems.
Moreover, the average initial utilization in the bottleneck resource is set around 80-90%, again to consider a more realistic situation.

- **Production line is unsaturated and job arrival in the line is a stochastic process**: Unsaturated lines are production lines where the system is limited by a stochastic arrival pattern, in contrast with saturated production lines, where starvation in the first stage can never occur, meaning that the system is not dependent on material supply or incoming demand.
  A stochastic process is a sequence of observations of the same random variable as it is observed through time. The arrival of the jobs in the line is hypothesized to follow a Poisson point process; this is a widely common assumption, since the Poisson distribution is the discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time if these events occur with a known constant mean rate and independently of the time since the last event. Thus, the inter-arrival time of the jobs follows an exponential distribution, which is the probability distribution of the time between events in a Poisson point process.

- **Job processing is a stochastic process**: Job processing time in resources is assumed to follow a lognormal distribution, which is a continuous probability distribution of a random variable whose logarithm is normally distributed.

- **Transfer time is negligible**: After being processed by a resource, jobs are sent to the buffer of the following stage. The job transfer neither use a vehicle or an operator nor requires time to complete. So, if the buffer in the following stage is not full, the resource is released, and the processed job moves directly and immediately in the next buffer.

- **No failures during resource process and no setups**: Breakdowns while the resources are processing jobs and maintenance do not occur in the considered production lines. This simplification is made to limit the sources of variability in the system to better focus on other variables.

- **Buffers with limited capacity**: Buffers capacity is limited, so when a job tries to leave a resource, if the following stage buffer is saturated, the resource is subject to blocking. An exception is the buffer in the first stage, which has never limited capacity to avoid job loss at the arrival. In chapter 6 also unlimited buffers have been briefly considered (even if not realistic), to observe the effects of blocking absence on queues. Indeed, if buffers have

infinite capacity, no blocking occurs and queues are influenced only by process capacity of the resource in the same stage where the buffer is placed, and by process capacity of the resource in the previous stage. Then, this simplified situation can be compared with the realistic one, where buffers are limited.

- **Blocking after service (BAS)**: BAS is a resource blocking mechanism which prescribes that, after a job has been processed, if the job tries to enter in the following queue which has reached its capacity constraint, then it is forced to wait in the resource, forbidding to process the next jobs. When a space becomes available in the following buffer, the job moves, the resource is released, and service can resume.
- **Unconstrained departures**: After being processed by the last stage resource, jobs leave the system freely, without any blocking mechanism.

2.4 Considered sensor positions

To build a full functioning DT many types of sensors can be involved. For example, in [citare Sensors 4.0 - Smart sensors and measurement technology enable Industry 4.0] an IoT application in a hydraulic system is described: the outputs of different kinds of sensors (e.g. pressure sensors, flow sensors, electrical power sensors, temperature sensors, vibration sensors) are combined to calculated indicators in order to identify hydraulic system faults and defects of the sensors themselves. The same information could be used to keep a hypothetical DT aligned with the real system in every subtle detail.
However, a process model, which is the fundamental structure of a DT, can be built requiring much less information. Indeed, to automatically create a basic process model using PM, even simple event logs are sufficient. This kind of event logs are obtainable as output of sensors, embedded in specific places along the line, that record just the ID of the case flowing through the sensor position and the timestamp corresponding to the passage instant. Therefore, given the simple task of the sensors considered in this application, not the sensor type, but the placement choice is critical.
In this thesis, a placement configuration with three sensors in each stage is assumed to be implemented:

1. One sensor at the buffer entrance. Since, as explained in [citare considered system], transfer time is assumed always null, this position also corresponds to the resource exit of the previous stage

2. One sensor at the resource entrance, corresponding to the buffer exit

3. One sensor in the resource, which records job processing finish. It is to be noted that it does not coincide with the resource exit: indeed, a job could stop inside the resource if the buffer in following stage is full, because of the BAS policy.

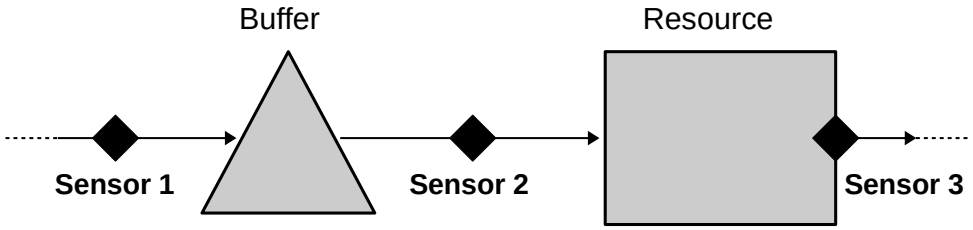


Fig. 2.5 Three sensors configuration scheme

All the sensors are identical, and they all collect the same data, that are the job identification code (Case ID) and its passage instant (Timestamp). These are not the only fields composing the output event log, other attributes are associated, but they are not job-related fields, which have to be actively recorded by the sensors (like Case ID and timestamp), instead they keep track of the event and the sensor positions.
At the end of the thesis the topic of which indicators are obtainable if only two sensors are present in each stage is discussed. This situation could occur because of the breakage of one sensor or in the case of an placement design that aims to reduce time and costs related to implementation.

2.5 Considered process variations

2.5.1 Variation classification

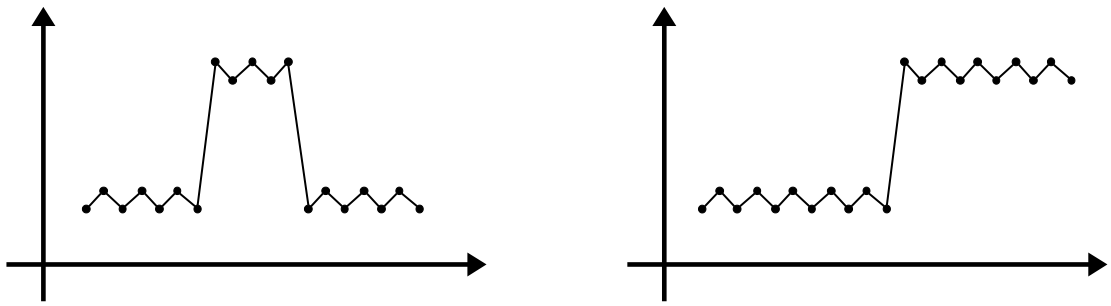
The problem of process variation has been addressed in Process Mining, in particular within the wider topic of Streaming Process Mining, that is the construction of models based not on finite event logs but on streams of events. Indeed, in the long term, an event stream relative to a process monitored in real time could be susceptible to concept drifts, which are “situations where the process is changing while being analyzed” [citare libro]. In [citare Handling concept drift in Process Mining] by Bose et al., concept drifts are classified based on three criteria.

Classification on perspective It focuses on where the change occurs, indicating not the physical location but the level of variation. Three categories are identified

- Control-flow perspective: this class includes structural changes, such as insertions, deletions, substitutions, and reordering of process components. For example, inserting new stages in the line, or adding new product types, cause variations of the model structure itself.
- Data perspective: this class includes changes related to requirements, usages, a data generation. For example, the enabling of performing a task without the request of information previously needed.
- Resource perspective: this class includes changes in roles and behaviors of resources. These variations influence the executions of a process, for example a resource capacity increase in the bottleneck can make the throughput increase if the process is unstable (i.e. production-constrained), or, in a multi-resource stage, a machine disabling makes the stage slower and raises its utilization.

Classification on duration As the name suggests, it aims to classify the variations based on their durations. Two categories are identified

- Momentary changes: variation lifespans are short and may affect few activities, being not enough durable to influence the whole process. (Figure 2.6a)
- Permanent changes: variations lifespans are long-lasting, to the point that can be considered permanent. These changes usually affect all the process activities. (Figure 2.6b)

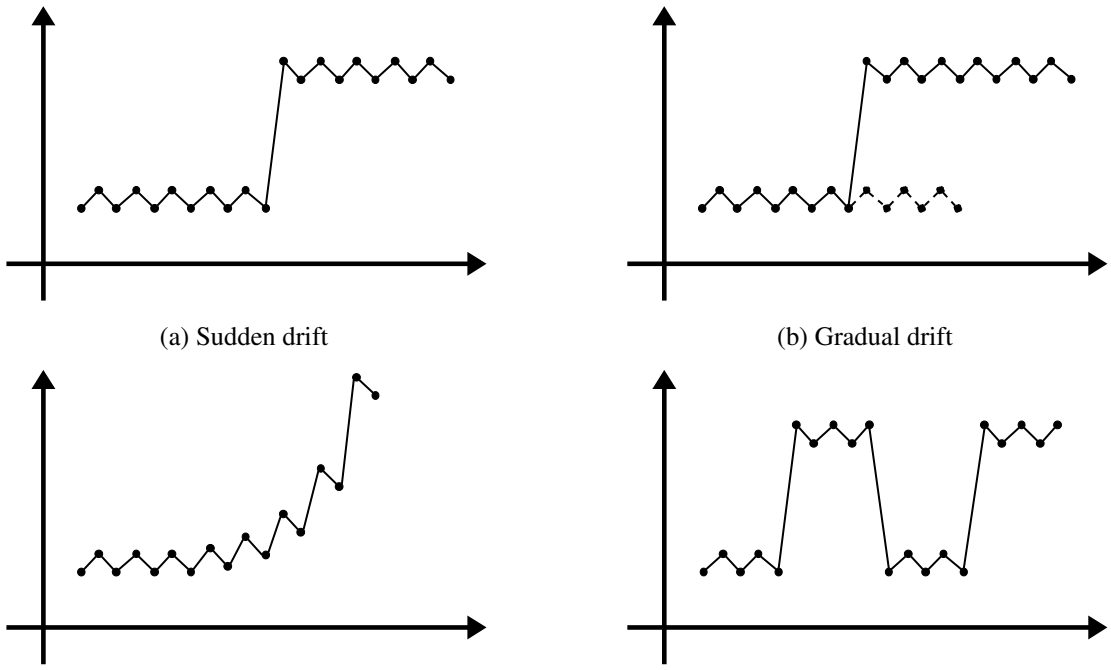


(a) Momentary change (b) Permanent change

Fig. 2.6 Change classification on duration

Classification on nature It focuses on the variation speed and frequency. Four categories are identified

- Sudden drift: it includes variations consisting of instantaneous substitution of an existing process with a new one. This kind of change typically occurs in emergency circumstances, such as machine failures. (Figure 2.7a)
- Gradual drift: it includes variations that occur without substituting the previous conditions, in other words it considers situations where different processes coexist for a certain time. This kind of variation is typical of planned organizational changes, such as the implementation of new procedures, without suddenly discontinuing the old ones. (Figure 2.7b)
- Incremental drift: it includes variations that occur through small and incremental steps. This is typical of processes with slow machine deterioration over time. (Figure 2.7c)
- Recurring drift: it includes variations which occur periodically, often with fixed frequency, followed by reinstatements of previous conditions. It is typical of processes characterized by seasonality. (Figure 2.7d)



(a) Sudden drift

(b) Gradual drift

(c) Incremental drift

(d) Permanent drift

Fig. 2.7 Change classification on nature

2.5.2 Considered variations

In this thesis two particular variation types are considered

- Processing time variation in a resource: a machine in a stage is subject to a sudden production capacity growth or loss, causing, respectively, a decrease or an increase of time needed to process a job in the stage. A capacity growth in a stage may occur in case of a machine substitution with a new and more efficient one. On the other side, a capacity loss may occur when an unnoticed resource deterioration is not adequately addressed by maintenance and abruptly affects the machine, slowing its production.
- Buffer capacity variation: a buffer in a stage is subject to a sudden capacity limit growth or loss. This kind of variation may occur in a plug-and-play context, where buffers can be expanded or shrink based on the stage requirements, aiming to minimize blocking and starvation along the line.

Therefore, according to the previous categorization, the variations considered in this dissertation are limited to stage-related, permanent, sudden changes. It is to be noted that the expression "stage-related" is used as the adaptation to a manufacturing context of the resource-perspective category, part of the classification on perspective. Indeed, in manufacturing the term resource is often used as synonym of machine, yet a resource-change, as meant above, can occur to any stage component, including buffers: a buffer capacity variation does not affect the model structure or the data requirement and usage, but influences stage behaviors and, so, the process execution, which is the distinctive trait of a resource-perspective variation.

Chapter 3

State of the art and new approach

Chapter 4

Indicator extractions from event logs

An event log, as described in [citare section event log], has a well-defined general structure, characterized by the association of one row to each event, and can include a variable number of attributes (i.e. fields). In this thesis event logs produced by the sensors are assumed to be composed of five fields

- **Event ID**: univocal code of the event. It is assumed that the event order sequence serves as ID of the event. Therefore Event ID works on an ordinal scale and events are represented with indexes belonging to a set Θ whose elements are $e \in \{1, 2, ..., E\}$, where E is number of the events occurring during the process. It is to be noted that the information carried by Event ID is redundant, as explained in section 4.1, so this field can be discharged without information loss.

- **Case ID**: univocal code of the job. Since the considered lines are SSLs, with one resource in each stage, and the service policy is BAS, as explained in [citare section considered systems], no case order swap is possible, so the Case ID is equivalent to the job arrival order in the first stage of the line and to the job processing order in each stage. The job arrival order is used as ID for the jobs. Therefore Case ID works on a categorical scale and jobs are represented with indexes belonging to a set Γ whose elements are $j \in \{1, 2, ..., J\}$, where J is the total number of arrived (and processed) jobs in the line.

- **Activity ID**: univocal code of the stage where the event occurs. It is to be noted that the definition of activity for the rest of the dissertation is slightly different from the one presented in [citare section event log]: the term activity is not used to indicate a step in the work sequence, but a station in the manufacturing line. Since lines are SSLs, and so

stages cannot be concurrent, the occupied position in the line stage sequence is used as identification code of the stage. Therefore, Activity ID works on a categorical scale and stages are represented with indexes belonging to a set Σ whose elements are $s \in \{1, 2, ..., S\}$, where S is the number of stages in the line.

- **Position ID**: univocal code of the position occupied by a sensor in a stage. It is to be noted that this ID is univocal only inside a certain stage: multiple sensors in different stages own the same ID if they occupy the same relative position. Position ID works on a categorical scale and positions are represented with indexes belonging to a set Π whose elements are $p \in \{1, 2, ..., P\}$, where P is the number of sensors in each stage, which depends on the considered sensor configuration [rinviare a sensor positions].

- **Timestamp**: instant when the event occurred. This field works on an interval scale, where the zero value is assumed in correspondence with the process starting instant.

4.1 From event logs to case lists

Before the indicator extractions, the event log is modified, moving from an event perspective to a case-activity perspective. The dataset obtained modifying event logs is named case list. This conversion is applied to simplify the successive calculations and to organize the data in a structure that better fits this stage-focused outlook.
A preliminary transformation is the discharge of Event ID. Indeed, both Timestamp and Event ID carry the same information (i.e. the event order), however they work on different measurement scales. Timestamp works on an interval scale, instead Event ID operates on an ordinal scale. Event ID clearly does not convey any information regarding the time gap between events, just their order of occurrence. Thus, Timestamp is kept, since it is the richer field, and Event ID is dismissed.
The rest of the event log undergoes a pivoting operation. Timestamp field is spread in three (or two, in a two-sensor design) fields, separating time instants acquired by sensors located in different positions (recorded in Position ID) in the same stage. Considering the three-sensors configuration, the resulting fields are named as follows

- **Timestamp_Buffer** T_B: it registers the instant when the job entered in the stage buffer

- **Timestamp_Resource** T_R: it registers the instant when the job entered in the stage resource

- **Timestamp_End** T_E: it registers the instant when the resource finished the job processing

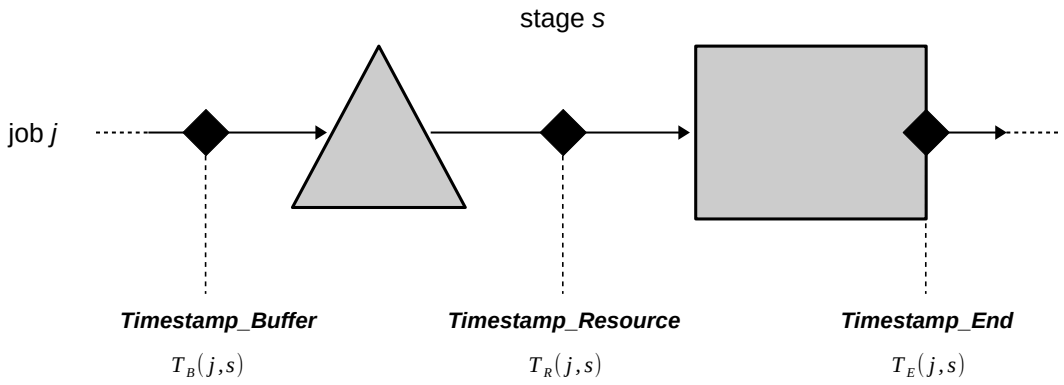


Fig. 4.1 Relations between sensors and timestamps

Each row of the resulting dataset records the instants when the sensors in a specific stage observed the passage of a certain job. Each row is uniquely identified by the combination of Case ID and Activity ID.
Therefore, Timestamp fields are referenced as

- $T_B(j,s)$: instant of entrance of job j in the buffer of stage s
- $T_R(j,s)$: instant of entrance of job j in the resource of stage s
- $T_E(j,s)$: instant of processing finish of job j in stage s

4.2 Indicator computations

Both basic and derived KPIs are calculated starting from case lists. In this section firstly the basic ones are presented, from which all the others are obtained. Then KPIs calculated aggregating on moving windows are discussed.

4.2.1 Basic KPIs

These indicators are computed as differences between timestamps, so they are all time intervals. They are separated in two categories: **Same Case Intervals** (**SCI**) KPIs and **Consecutive Cases Intervals** (**CCI**) KPIs. The first category includes indicators calculated as differences of timestamps related to the specific job, not necessarily in the same stage. The second category

consists of indicators calculated as differences of timestamps related to pairs of jobs consecutively passing in the same sensor position.

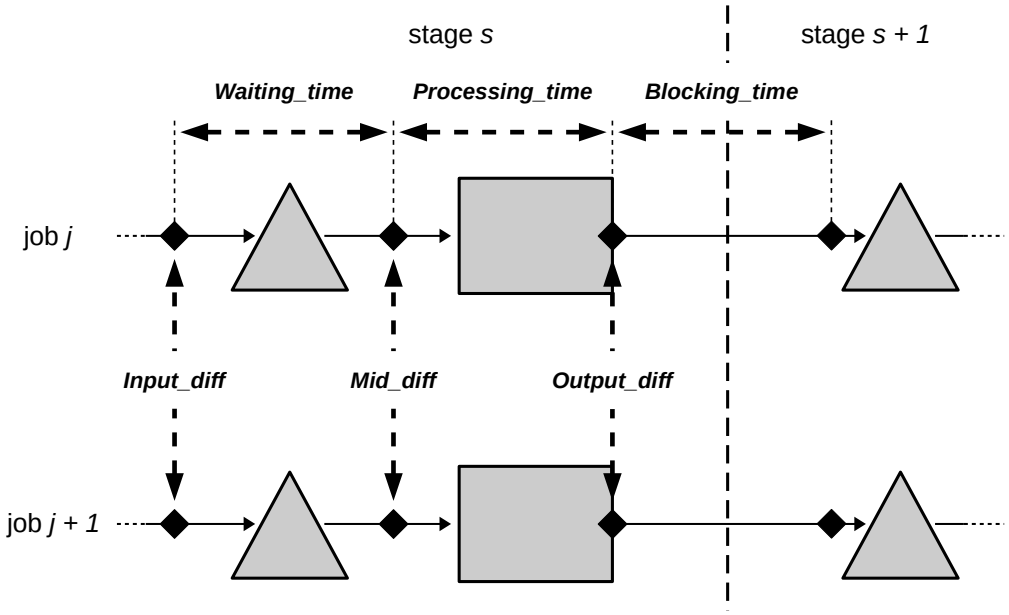


Fig. 4.2 Basic KPIs scheme

Waiting_time is a SCI indicator, computed as the difference between Timestamp_RESOURCE and Timestamp_BUFFER. Since Timestamp_BUFFER is the instant when the job entered in the stage buffer, and Timestamp_RESOURCE is the instant when the job departed from the buffer and entered in the stage resource, this KPI represents the time spent by a job in a stage buffer. The equation for Waiting_time of job j in stage s is

$$Waiting\_time(j,s) = T_R(j,s) - T_B(j,s)$$

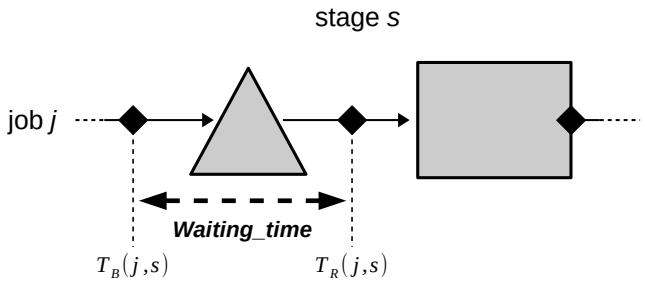


Fig. 4.3 Waiting time scheme

Processing_time is a SCI indicator, computed as the difference between Timestamp_END and Timestamp_RESOURCE. Since Timestamp_RESOURCE is the instant when the job entered in the stage resource (and started being processed), and Timestamp_END is the instant when the job processing finished, this KPI represents the time spent by a job in a stage resource while being processed.
The equation for Processing_time of job j in stage s is

$$Processing\_time(j,s) = T_E(j,s) - T_R(j,s)$$

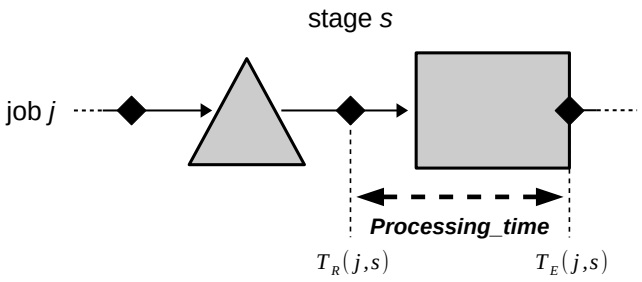


Fig. 4.4 Processing time scheme

Blocking_time is a SCI indicator, computed as the difference between Timestamp_BUFFER, registered at the entrance of the stage following the considered one, and Timestamp_END, registered in the considered stage. Since Timestamp_END is the instant when the job ended to be processed by the stage resource, and Timestamp_BUFFER is the instant when the job entered in the buffer of the following stage, this KPI represents the time spent by a job in a stage resource not being processed; in other words, this is the time interval during which the resource was occupied by the job because the buffer in the following stage was full, causing the block of stage s resource.
The equation for Blocking_time of job j in stage s is

$$Blocking\_time(j,s) = T_B(j,s+1) - T_E(j,s)$$

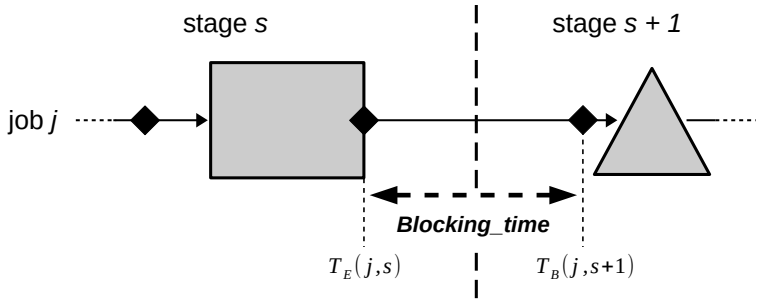


Fig. 4.5 Blocking time scheme

Input_diff, Mid_diff, and Output_diff are CCI indicators, computed as the differences between pairs of timestamps (respectively pairs of Timestamp_BUFFER, pairs of Timestamp_RESOURCE, and pairs of Timestamp_END) relative to consecutive job passages through the same sensor. All these KPIs represent time intervals related with the cycle time as perceived by different sensor positions; the values assumed by these indicators during a process in steady state are very similar, as it is going to be shown in subsection 6.1.1.
The equations for CCI KPIs of job j in stage s are

$$Input\_diff(j,s) = T_B(j+1,s) - T_B(j,s)$$

$$Mid\_diff(j,s) = T_R(j+1,s) - T_R(j,s)$$

$$Output\_diff(j,s) = T_E(j+1,s) - T_E(j,s)$$

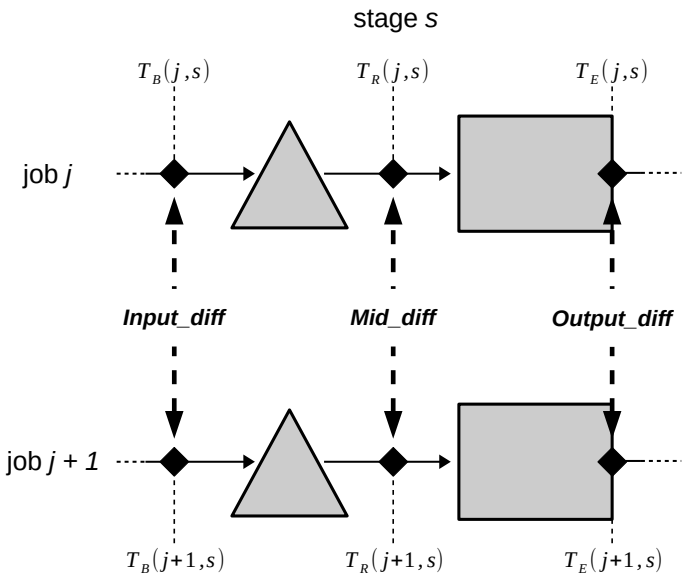


Fig. 4.6 Consecutive cases intervals scheme

4.2.2 Derived KPIs

These KPIs are obtained combining the basic KPIs introduced in the previous paragraph. The first indicator belonging to this group is called Starving_time and is closely related the SCI KPIs, to such an extent that is considered part of that category for the rest of the dissertation. The remaining derived KPIs are classified in two categories: Trend KPIs and Rolling Windows KPIs.

Starving_time is calculated as the difference between Mid_diff and the sum of Processing_time and Blocking_time. As said, Mid_diff represents the time interval between the passages of two consecutive jobs through the same sensor; this means that Mid_diff cannot be less than Processing_time (that is, the time spent by a job being processed), since there is only one resource in each stage and, if the resource is seized, jobs have to wait in the buffer queue. Moreover, Mid_diff can be greater than Processing_time for two reasons: firstly, if the buffer of the following stage is full, a processed job does not release the resource, causing the machine blocking (for a period represented by Blocking_time) and preventing other jobs to enter in the resource. Secondly, after a job has released the resource, if the previous buffer is empty, clearly no job seizes the resource, causing starvation. Both these situations contribute to inflate Mid_diff, that is none other than the sum of these time intervals plus the processing time. Therefore, knowing Mid_diff, Processing_time, and Blocking_time, Starving_time can be calculated and represents

the time interval during which a resource was not working due to lack of jobs in the stage buffer. The equation for Starving_time of job j in stage s is

$$Starving\_time(j,s) = Mid\_diff(j,s) - (Processing\_time(j,s) + Blocking\_time(j,s))$$

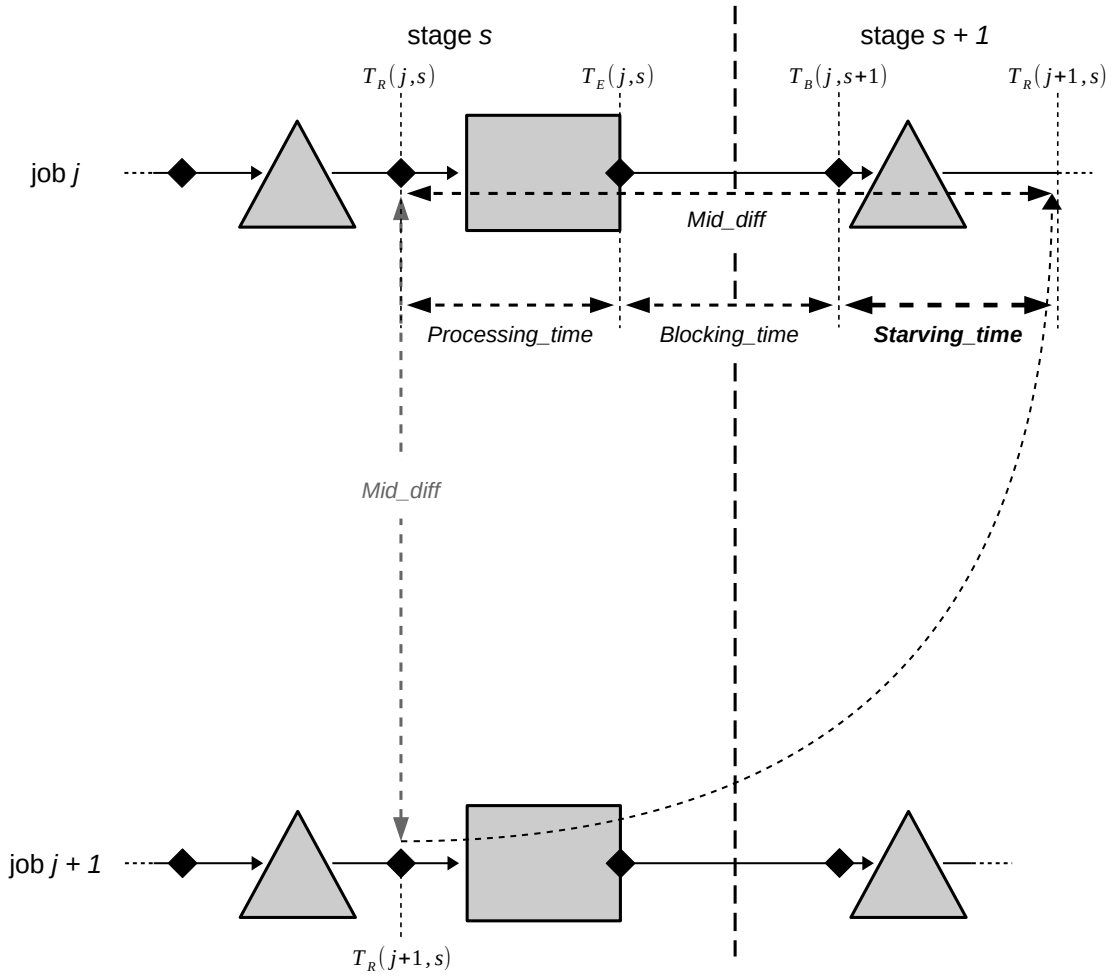


Fig. 4.7 Starving time scheme

A more direct, yet less intuitive way to compute this KPI (found simply by replacing the basic indicators with the equations that define them) is differentiating Timestamp_RESOURCE of the job, immediately following the considered one, entering in the stage resource, and Timestamp_BUFFER of the considered job entering in the next stage buffer.
So, an alternative formula to compute Starving_time of job j in stage s is

$$Starving\_time(j,s) = T_R(j+1,s) - T_B(j,s+1)$$

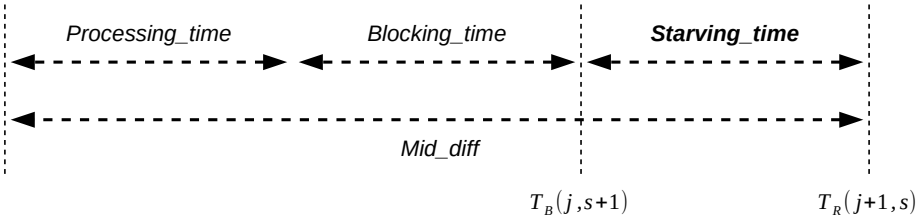


Fig. 4.8 Starving time formula visualization

Trend KPIs

Trend KPIs are calculated as ratios of CCI KPIs. Since both numerator and denominator are time intervals, these KPIs present as absolute values. They aim to display how the job flow in a specific stage portion behaves in time.

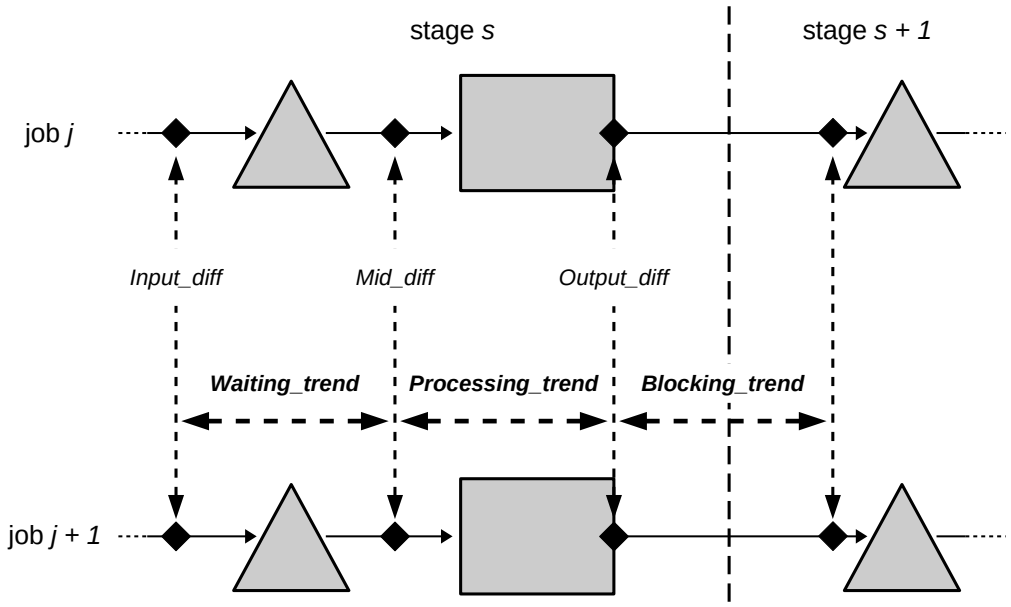


Fig. 4.9 Trend KPIs scheme

Waiting_trend is computed as the ratio of Mid_diff and Input_diff both referring to the same job in the same stage. Since Mid_diff represents the flow time at a buffer exit, and Input_diff represents the flow time at a buffer entrance, this KPI shows if the buffer outflow is higher than the inflow or vice versa. Therefore, a greater than one Waiting_trend (corresponding to a higher outgoing than ingoing flow time, that is a lower outgoing than ingoing flow rate) suggests that the buffer is filling up, whereas a lower than one Waiting_trend (corresponding to a higher ingoing

than outgoing flow time, that is a lower ingoing than outgoing flow rate) suggests that the buffer is emptying.
The equation for Waiting_trend of job j in stage s is

$$Waiting\_trend(j,s) = \frac{Mid\_diff(j,s)}{Input\_diff(j,s)}$$

Processing_trend is computed as the ratio of Output_diff and Mid_diff referring to the same job in the same stage. Since Output_diff represents the flow time in correspondence with a job processing finish, and Mid_diff represents the flow time at a resource entrance, this KPI shows if the resource outflow is higher than the inflow or vice versa. Therefore, a greater than one Processing_trend (corresponding to a higher outgoing than ingoing flow time, that is a lower outgoing than ingoing flow rate) suggests that the resource is slowing down (i.e. its process capacity is reducing), whereas a lower than one Processing_trend (corresponding to a higher ingoing than outgoing flow time, that is a lower ingoing than outgoing flow rate) suggests that the resource is accelerating (i.e. its process capacity is increasing).
The equation for Processing_trend of job j in stage s is

$$Processing\_trend(j,s) = \frac{Output\_diff(j,s)}{Mid\_diff(j,s)}$$

Blocking_trend is computed as the ratio of Input_diff of a job in the stage following the considered one, and Output_diff of the same job in the considered stage. Since Input_diff represents the flow time at a buffer entrance, and Output_diff represents the flow time in correspondence with a job processing finish, this KPI shows how blocking affects job flow. Therefore, a greater than one Blocking_trend suggests that blocking time is increasing (i.e. resource is subject to longer blocking times), whereas a lower than one Blocking_trend suggests that blocking time is decreasing (i.e. resource is subject to lower blocking times). These indicators are quite useful, even if the carried insights are similar to the ones already offered by corresponding SCI KPIs. Indeed, as [sezione results] shows, these Trend KPIs are steadier and less noise affected, but more reactive to changes than SCI KPIs, resulting suitable to quickly detect variations in the process behavior.
The equation for Blocking_trend of job j in stage s is

$$Blocking\_trend(j,s) = \frac{Input\_diff(j,s+1)}{Output\_diff(j,s)}$$

Rolling Windows KPIs

Rolling Windows (RW) KPIs are indicators calculated on case list subsection. Since this operation aggregates groups of consecutive rows (i.e. windows) in single records, the resulting dataset has as fewer rows as the subsections are wider. The table containing the aggregation results is named aggregated case list.
RW KPIs are separated in two categories: Moment RW KPIs and Stage Metric RW KPIs. The behaviour of this kind of indicators deeply depends on the characteristics of the window which they are computed on. Indeed, windows are characterized by two parameters: length and shift. The length l defines how many rows (i.e. jobs) are included in the dataset subsection. The shift f defines how many rows are between the start of a subsection and the start of the following one, upper extreme included. So, windows overlap if shift is lower than length, windows are delayed if shift is higher than length, windows are adjacent if the parameters are equal. Only complete subsections are considered (i.e. all the windows have length l, included the first and the last ones), and it is to be noted that the number of windows obtained from a case list varies in function of the windows parameters: the higher the shift and the length, the lower the number of windows obtained from a case list and, therefore, the fewer the rows of the aggregated case list. Since each window includes l jobs, it is conventionally assumed that the Case ID value of the last job of a window serves as window ID. So, windows are represented with a set Ω, a subset of set Γ, whose elements $w_{l,f}$ depend on window parameters l and f, and that includes only indexes of jobs positioned at window ends. So, given certain window parameters values, the elements of set Ω are defined through the function $w_{l,f} \in \{l, l+f, l+2f, ...\} \subseteq \Gamma$. Considering that, for the rest of the thesis the function $window(w_{l,f}, l)$ is used to indicate the window ending with job $w_{l,f}$ and having length l.

Moment Rolling Windows KPIs are RW indicators related to moments computed on case list subsections. First order moment (i.e. mean) and second order central moment (i.e. variance) are calculated for each SCI (Starving_time included), CCI, and Trend KPI. The third order standardized moment (i.e. skewness, obtained standardizing the third order central moment) is computed only on Waiting_time KPIs.
These indicators aim to reduce the noise due to process variability, and to concisely display the characteristics of the distributions followed by the related KPIs.
The equations for Moment Rolling RW KPIs are

$$Waiting\_time\_MEAN(w_{l,f}, l, s) = \text{mean}_j\{Waiting\_time(j, s)\}, j \in \{w_{l,f} - l + 1, w_{l,f} - l + 2, ..., w_{l,f}\}$$

$$Waiting\_time\_VAR(w_{l,f}, l, s)$$

$$Waiting\_time\_SKEW(w_{l,f}, l, s)$$

$$Processing\_time\_MEAN(w_{l,f}, l, s)$$

$$Processing\_time\_VAR(w_{l,f}, l, s)$$

$$Blocking\_time\_MEAN(w_{l,f}, l, s)$$

$$Blocking\_time\_VAR(w_{l,f}, l, s)$$

$$Starving\_time\_MEAN(w_{l,f}, l, s)$$

$$Starving\_time\_VAR(w_{l,f}, l, s)$$

$$Input\_diff\_MEAN(w_{l,f}, l, s)$$

$$Input\_diff\_VAR(w_{l,f}, l, s)$$

$$Mid\_diff\_MEAN(w_{l,f}, l, s)$$

$$Mid\_diff\_VAR(w_{l,f}, l, s)$$

$$Output\_diff\_MEAN(w_{l,f}, l, s)$$

$$Output\_diff\_VAR(w_{l,f}, l, s)$$

$$Waiting\_trend\_MEAN(w_{l,f}, l, s)$$

$$Waiting\_trend\_VAR(w_{l,f}, l, s)$$

$$Processing\_trend\_MEAN(w_{l,f}, l, s)$$

$$Processing\_trend\_VAR(w_{l,f}, l, s)$$

$$Blocking\_trend\_MEAN(w_{l,f}, l, s)$$

$$Blocking\_trend\_VAR(w_{l,f}, l, s)$$

Stage State Rolling Windows KPIs are RW indicators that help to monitor buffer and resource, exploiting the flow time to get more insights about their usage. Concerning the buffer, the average number of jobs (in a certain window) in a queue is calculated; then three resource related metrics are computed, that are utilization, probability of blocking, and probability of starvation. All Stage State RW KPIs present the same formula structure, that is the ratio between a SCI derived Moment RW KPI and a CCI derived Moment RW KPI, both calculated on the same dataset subsection. Since both numerator and denominator are time intervals, these KPIs present as absolute values.

Average_Queue is calculated as the ratio between the Waiting_time_MEAN and Input_diff_MEAN. Since Waiting_time_MEAN is the average time interval spent by jobs waiting in a buffer, and Input_diff_MEAN is the average time interval between consecutive jobs entering in a buffer (i.e. average flow time as seen by the buffer), their ratio tells the average number of jobs simultaneously present in a buffer. As said in
The equation for Average_Queue computed on window $w_{l,f}$ in stage s is

$$Average\_Queue(w_{l,f}, l, s) = \frac{Waiting\_time\_MEAN(w_{l,f}, l, s)}{Input\_diff\_MEAN(w_{l,f}, l, s)}$$

Utilization is calculated as the ratio between the Processing_time_MEAN and Mid_diff_MEAN. Since Processing_time_MEAN is the average time interval needed by a resource to process a job, and Mid_diff_MEAN is the average time interval between consecutive jobs entering in a resource (i.e. average flow time as seen by the resource), their ratio tells how much time the resource worked compared to the available time, or, from another point of view, the probability that, in a certain instant, the resource was not idle.
The equation for Utilization computed on window $w_{l,f}$ in stage s is

$$Utilization(w_{l,f}, l, s) = \frac{Processing\_time\_MEAN(w_{l,f}, l, s)}{Mid\_diff\_MEAN(w_{l,f}, l, s)}$$

Blocking_probability is calculated as the ratio between the Blocking_time_MEAN and Mid_diff_MEAN. Since Blocking_time_MEAN is the average duration of a stop when blocking occurs in a resource, and Mid_diff_MEAN is the average time interval between consecutive jobs entering in a resource (i.e. average flow time as seen by the resource), their ratio tells how much time the resource was in a blocking state compared to the available time, or, from another point of view, the probability that, in a certain instant, the resource was blocked.
The mathematical equation for Blocking_probability computed on window $w_{l,f}$ in stage s is

$$Blocking\_probability(w_{l,f},l,s) = \frac{Blocking\_time\_MEAN(w_{l,f},l,s)}{Mid\_diff\_MEAN(w_{l,f},l,s)}$$

Starving_probability is calculated as the ratio between the Starving_time_MEAN and Mid_diff_MEAN. Since Starving_time_MEAN is the average time a resource has to wait for the arrival of a new job after being released by the previous one, and Mid_diff_MEAN is the average time interval between consecutive jobs entering in a resource (i.e. average flow time as seen by the resource), their ratio tells how much time the resource was in a starving state compared to the available time, or, from another point of view, the probability that, in a certain instant, the resource was starving.
The equation for Starving_probability computed on window $w_{l,f}$ in stage s is

$$Starving\_probability(w_{l,f},l,s) = \frac{Starving\_time\_MEAN(w_{l,f},l,s)}{Mid\_diff\_MEAN(w_{l,f},l,s)}$$

Chapter 5

Numerical experiments

To observe the indicator behaviours, simulations are built to replicate a simple serial line having the characteristics described in section 2.3. Virtual sensors have been placed accordingly to the three sensor configuration defined in section 2.4. The sensor output is an event log having the field structure described in chapter 4.
Concerning the software tools:

- Models are written and run with *simmer* [citare simmer]
- Event logs transformation in case lists and KPI extractions are executed with queries mostly written with *dplyr*, *tidyr* and other libraries belonging to the *tidyverse* collection [citare tidyverse]
- Plots are printed using *ggplot2*, which is also part of the *tidyverse* collection.

simmer and *tidyverse* are open-source libraries for R language.

5.1 Simulated line

The model is designed to simulate a simple serial line having 7 stages. Each stage is composed by a resource fed by a buffer. The first buffer is supplied by a source set up to generate 10000 jobs; job inter-arrival time (i.e. inter-generation time) follows an exponential distribution with

pdf

$$f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x}, & \text{if } x \geq 0 \\ 0, & \text{otherwise.} \end{cases} \tag{5.1}$$

having fixed parameter $\lambda = 10^{-1}$. In this way, the expected value of the inter-arrival time is $\mu_a = 10$ time units. The simulation stops when the last job has been processed by the resource in the last stage.
Job processing time in the resources follows a log-normal distribution with pdf

$$f(x;\mu,\sigma^2) = e^{\mu+\sigma Z(x;\mu,\sigma^2)} \tag{5.2}$$

where $Z(x;\mu,\sigma^2)$ is the pdf of a normal distribution with mean μ and variance σ^2. The standard deviation of the log-normal distribution is set to $\sigma_s = 5$ time units in every stage s; the initial mean is set to $\mu'_s = 7$ time units in every stage, except for the resource in the bottleneck stage s_{BN}, where the initial mean processing time is set to $\mu'_s = 8.5$.
During the simulation run, a variation takes place in the line, to observe how indicators behave when it occurs. The variation is realized forcing a process parameter to suddenly change in the instant when half of the jobs (i.e. the 5000 jobs) have been processed in the first stage buffer. The changing stage s_{CH} is chosen to have a wide view of the variation effects both upstream and downstream in the line; the change trigger was set to be sure that system was in steady state before the variation and to let it reach the steady state again after.

5.1.1 Experimental factors and levels

In this section the experimental factors and the respective levels are presented.

- **Bottleneck and changing stage positions** Bottleneck and changing stage positions in a model depend on each other. Three stage configurations are considered to observe KPIs in different situations, that are

  1. Bottleneck stage s_{BN} is upstream the changing stage s_{CH}
  2. Bottleneck stage s_{BN} is in correspondence with the changing stage s_{CH}
  3. Bottleneck stage s_{BN} is downstream the changing stage s_{CH}

So, the stage configuration pairs are

$$(s_{BN}, s_{CH}) \in \{(3,5),(4,4),(5,3)\}$$

- **Initial buffer capacities** In each model, capacity limit cl_s of stage s buffer has the same initial value cl'_s in all stages. Four initial buffer capacity levels are considered

  1. Buffer capacity limit equal to 3 jobs: in this configurations buffers are small and easily saturated
  2. Buffer capacity limit equal to 6 jobs: in this configurations buffers are large and not commonly saturated
  3. Buffer capacity limit equal to 10 jobs: in this configurations buffers are very large and almost never saturated
  4. Unlimited buffers; it is to be noted that models with unlimited buffers have not been taken into account in case of buffer limit variations.

  Therefore, the before-change considered levels of cl_s are (expressed in job units)

$$cl'_s \in \{3,6,10,\infty\}$$

The variation types are classified in two categories, in turn divided in two sub-categories. The considered change sizes depend on bottleneck position or buffer capacity of the model. Therefore, for each variation subcategory, different cl_s and (s_{BN}, s_{CH}) entail different variation levels.

Processing time variation This type of variation is realized changing the parameter μ_s in the processing time distribution of the resource in the fourth stage. The levels of the after-change value μ''_s taken by parameter μ_s depend on the bottleneck position in the model. A scheme of mean processing time variations is contained in Table 5.1.

Processing time increase

- Bottleneck in third stage and variation in fifth stage or vice versa ($(s_{BN}, s_{CH}) = (3,5) \vee (5,3)$): considered mean processing time after-change values are (expressed in time units)

$$\mu''_s \in \{8, 9.5, 11\} \qquad s = s_{CH}$$

- Bottleneck and variation in same stage ($(s_{BN}, s_{CH}) = (4,4)$): considered mean processing time after-change values are (expressed in time units)

$$\mu_s'' \in \{9.5, 11\} \qquad s = s_{CH}$$

Processing time decrease

- Bottleneck in third stage and variation in fifth stage or vice versa ($(s_{BN}, s_{CH}) = (3,5) \vee (5,3)$): considered mean processing time after-change values are (expressed in time units)

$$\mu_s'' \in \{4.5, 6\} \qquad s = s_{CH}$$

- Bottleneck and variation in same stage ($(s_{BN}, s_{CH}) = (4,4)$): considered mean processing time after-change values are (expressed in time units)

$$\mu_s'' \in \{4.5, 6, 7.5\} \qquad s = s_{CH}$$

Buffer capacity variation This type of variation is realized changing the capacity limit cl_s of the changing stage s_{CH} buffer. The levels of the after-change capacity buffer cl_s'' depend on initial buffer capacity cl_s' present in the model. A scheme of buffer limit variations is contained in Table 5.2.

Buffer capacity increase

- Initial buffer capacity equal to 3 jobs ($cl_s' = 3$): considered buffer capacity after-change values (expressed in job units) are

$$cl_s'' \in \{6, 10, 20\} \qquad s = s_{CH}$$

- Initial buffer capacity equal to 6 jobs ($cl_s' = 6$): considered buffer capacity after-change values (expressed in job units) are

$$cl_s'' \in \{10, 20\} \qquad s = s_{CH}$$

- Initial buffer capacity equal to 10 jobs ($cl'_s = 10$): considered buffer capacity after-change values (expressed in job units) are

$$cl''_s \in \{20\} \qquad s = s_{CH}$$

Buffer capacity decrease

- Initial buffer capacity equal to 3 jobs ($cl'_s = 3$): considered buffer capacity after-change values (expressed in job units) are

$$cl''_s \in \{1\} \qquad s = s_{CH}$$

- Initial buffer capacity equal to 6 jobs ($cl'_s = 6$): considered buffer capacity after-change values (expressed in job units) are

$$cl''_s \in \{1,3\} \qquad s = s_{CH}$$

- Initial buffer capacity equal to 10 jobs ($cl'_s = 10$): considered buffer capacity after-change values (expressed in job units) are

$$cl''_s \in \{1,3,6\} \qquad s = s_{CH}$$

Each factor level combination is used to build a model which is replicated 30 times, in order to assure statistical validity and accuracy in results. Therefore, every model outputs 30 event logs, that are aggregated to calculate the average Timestamp on each combination of Case ID, Activity ID and Position ID.

Table 5.1 Average processing time variation levels

		Stage configuration (s_{BN}, s_{CH})	
		$(4,4)$	$(3,5) \vee (5,3)$
Buffer capacity levels cl'_s		$\{3,6,10,\infty\}$	
Average before-change processing time in changing stage $\mu'_{s_{CH}}$		8.5	7
Average after-change processing time in changing stage $\mu''_{s_{CH}}$	Increase $\Uparrow$	$\{9.5,11\}$	$\{8,9.5,11\}$
	Decrease $\Downarrow$	$\{4.5,6,7.5\}$	$\{4.5,6\}$

Table 5.2 Buffer capacity variation levels

		Initial buffer capacity cl'_s		
		3	6	10
Stage configuration (s_{BN}, s_{CH})		$\{(3,5),(4,4),(5,3)\}$		
Average after-change buffer capacity in changing stage cl''_s	Increase $\Uparrow$	$\{6,10,20\}$	$\{10,20\}$	$\{20\}$
	Decrease $\Downarrow$	$\{1\}$	$\{1,3\}$	$\{1,3,6\}$

5.1.2 Custom simulations

While performing tests conforming to the DOE, it was evident that some particular cases were not included. To show these situation, in chapter 6 some custom simulations have been inserted in the result dissertation.

5.2 Rolling windows parameters

5.3 Plot structure

Each KPI described in [citare paragrafo KPI] is plotted to display the relative indicator behavior. The graphs present all the same structure, showing the KPI value as a function of Case ID. In Processing_time plots, Processing_time_MEAN plots, CCI plots and CCI MEAN plots, the average inter-arrival time is displayed with a dashed line in correspondence with the value 10.

Chapter 6

Results

In this chapter the most meaningful results are presented and analyzed. Basic KPIs derived plots are not included since the noise makes them barely understandable, so only RW KPIs are shown and commented. Unless otherwise specified, the RW KPIs are computed windows with parameters $f = 100$ and $l = 100$.

6.1 Processing time variation

A processing time variation always causes a change in Processing_time basic and derived KPIs in correspondence with the changing stage, however, it has other effects on the stages upstream and downstream. In this section the most information-rich indicators are analyzed, starting from consecutive case intervals KPIs and their relationship with cycle time. Then, it is shown how processing time variations are directly displayed by Processing_time. After that, the complex behavior of Blocking_time and Starving_time is described. Lastly, Waiting_time and its suitability to show variations in queue lengths are presented.

6.1.1 Consecutive cases intervals KPIs

Consecutive cases intervals (CCI) KPIs are indicators strongly related to the cycle time. First of all, the behavior of one of them, Mid_diff, is shown in a processing time increase situation. Then, the stage synchronization caused by the cycle time propagation through the line is displayed

with this KPI. Finally, it is shown how similar are the behaviors of Input_diff and Output_diff with respect to Mid_diff.

Processing time increase effects on Mid_diff

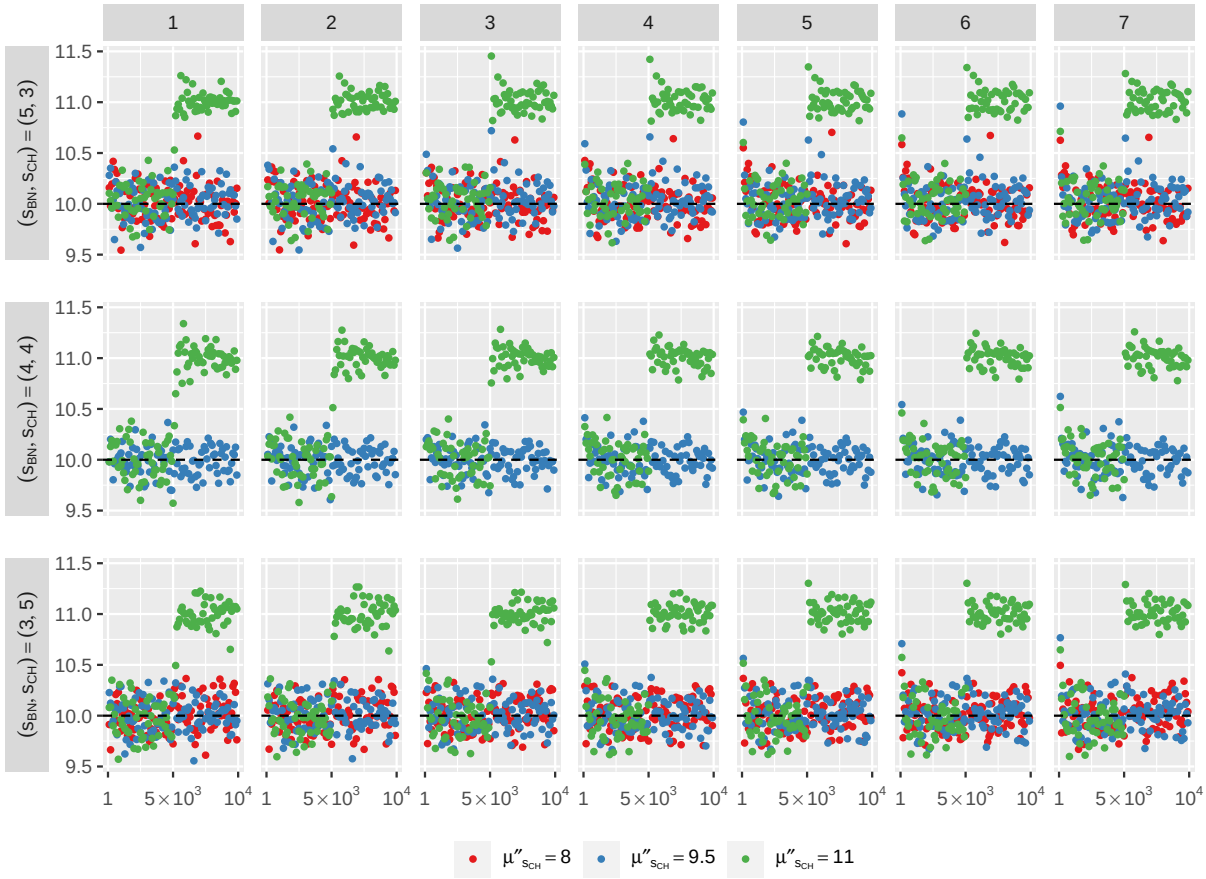


Fig. 6.1 Mid diff RW KPI considering three processing time increase levels $\mu''_{s_{CH}}$ and three bottleneck - changing stage configurations (s_{BN}, s_{CH}). Models with $cl_s = 6$.

Figure 6.1 shows Mid_diff_MEAN RW KPI behavior when a processing time growth occurs[1]

- Before the variation, the system is stable and average Mid_diff is around the average inter-arrival time μ_a.
  This is visible in all the models until Case ID reaches value 5000.

[1] In each plot average inter-arrival time is represented with a dashed line

- If a resource loses capacity, but the system remains stable (i.e. $\mu''_{s_{CH}} < \mu_a$) average Mid_diff does not change and remains around the average inter-arrival time μ_a.
  This is visible in models with $\mu''_{s_{CH}} = 8$ and $\mu''_{s_{CH}} = 9.5$ after Case ID has reached value 5000.

- If a resource loses capacity to the point that the system becomes unstable (i.e. $\mu''_{s_{CH}} > \mu_a$), average Mid_diff increases and aligns with the bottleneck average processing time $\mu''_{s_{CH}}$.
  This is visible in model with $\mu''_{s_{CH}} = 11$ after Case ID has reached value 5000.

- Mid_diff does not to depend on the stage position with respect to the changing stage or the bottleneck: it behaves similarly in all stages.

Mid_diff follows the cycle time behavior as it is described in theory: it stands around a value equal to the maximum between the bottleneck average processing time $\mu_{s_{BN}}$ and the average inter-arrival time μ_a[2].

Mid_diff propagation along the line

Even if it is scarcely visible in figure 6.1, Mid_diff does not immediately change in all stages, exactly like system cycle time: when the system becomes unstable, the new cycle time needs a certain time span to spread in stages both upstream, through blocking, and downstream, through starvation. Time required to re-establish process synchronization across the line depends mainly on current saturation of buffers: the more the buffers upstream are unsaturated, the longer cycle time takes to propagate upstream (blocking is delayed). The opposite occurs for downstream stages, where cycle time spread is slower the more the buffers are saturated (starvation is delayed).

[2]Actually, cycle time is equal to the maximum between the average inter-arrival time, average bottleneck processing time and average inter-departure time from the last stage, but since job departures from the system are assumed unconstrained in this thesis, the last term is always equal to 0 and can be omitted.

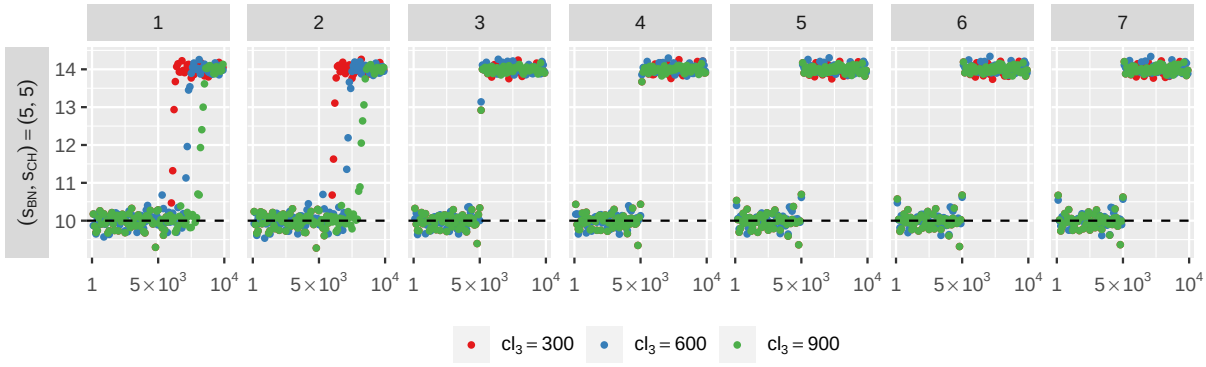


Fig. 6.2 Mid diff RW KPI variation delays considering different buffer capacity limits cl_3. Models with configuration $(s_{BN}, s_{CH}) = (5,3)$ and processing time increase $\mu''_{s_{CH}} = 14$.

Figure 6.2 better displays what just explained: the graphs portray a particular system where a single buffer having high capacity limit, placed in stage $s = 3$, is extremely unsaturated before the variation occurs. After the first stage has processed 5000 jobs, the resource in the bottleneck stage $s = 5$, positioned downstream stage $s = 3$, looses production capacity and its processing time increases to the point that it exceeds inter-arrival time and the system becomes unstable. Therefore cycle time increases, aligning with the new bottleneck average processing time, but the high capacity buffer works as a temporary decoupling point, delaying the blocking and, so, the synchronization of stages upstream its position. Moreover, these plots show that the bigger the buffer capacity (cl_3), the wider is the time span before cycle time (i.e. Mid_diff) manages to spread upstream that stage.
This means that, a variation that has effects on CCI KPIs is immediately detectable only in correspondence of the changing stage, while in the rest of the stages the variation could be delayed due to low (or high, when cycle time spreads downstream) buffer saturation levels.

Input_diff and Output_diff behaviors

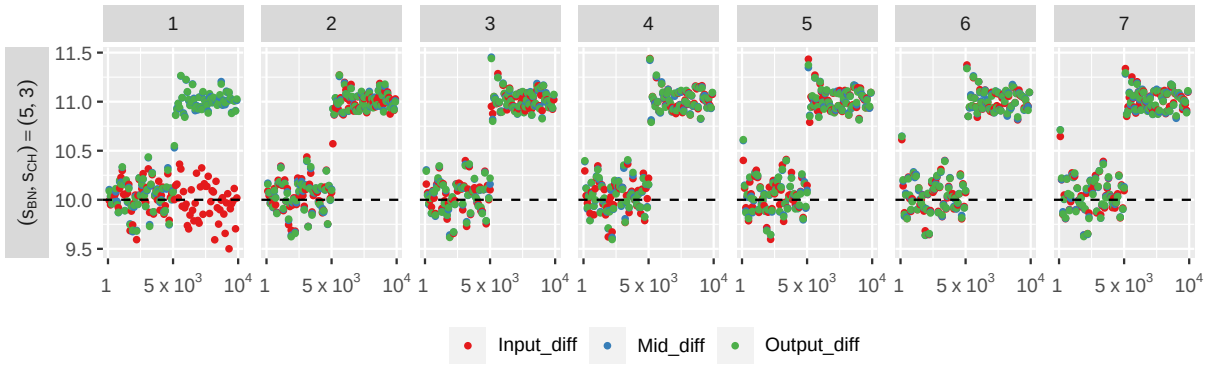


Fig. 6.3 Different CCI RW KPIs behaviors compared. Models with configuration $(s_{BN}, s_{CH}) = (4,4)$, $cl_s = 6$, and $\mu''_{s_{CH}} = 11$.

Input_diff and Output_diff behave similarly to Mid_diff, as figure 6.3 shows. Sure enough, CCI KPIs are all indicators of cycle time as it is seen in different stage positions. However, these KPIs are not exactly equal, and Input_diff and Mid_diff are not interchangeable when it comes to compute Starving time and Stage State RW KPIs: since Input_diff is relative to buffer entrances, it becomes equal to Mid_diff, which instead is relative to buffer exits, only when the buffer is saturated and the previous resource starts to suffer of blocking. So, in general, it cannot be used to calculate the exact Starving_time and cannot serve as denominator in Utilization, Blocking_prob and Starving_prob.
Values taken by CCI KPIs are very similar, but these KPIs differ from each other for little delays in their behaviors and generally are not equal.

Input_diff in the first stage

Since the lines considered in this thesis always have an unlimited buffer in the first stage, Input_diff allows to know the average inter-arrival time even when the system becomes unstable. As a matter of fact, the first buffer completely isolates the line entrance from potential system instabilities, and so Input_diff of the first stage is actually a permanent indicator of inter-arrival time, rather than an indicator of cycle time. This detail is visible in the the first stage plot of figure 6.3: after the variation, Mid_diff and Output_diff increase, while Input_diff does not change and remains equal to the average arrival time.

CCI KPIs usefulness

CCI KPIs are reliable indicators of cycle time and, therefore, can be used to understand if, after a variation, the system is still stable. However, since they change their behavior exclusively when a variation makes the system unstable, it is impossible to detect a decrease in processing time (i.e. a production capacity gain) monitoring them because this variation never causes system instability. Input_diff in the first stage has a unique behavior and, since the first stage buffer is unlimited, it is always an inter-arrival time indicator.

6.1.2 Processing_time and Utilization KPIs

Processing_time is the most useful indicator to understand if a resource production capacity has changed and to determine its new value. In this subsection, Processing_time behavior is analyzed in situations of processing time increase and decrease. Then, Utilization KPI is presented and its relation with Processing_time is displayed.

Processing time increase effects on Processing_time KPI

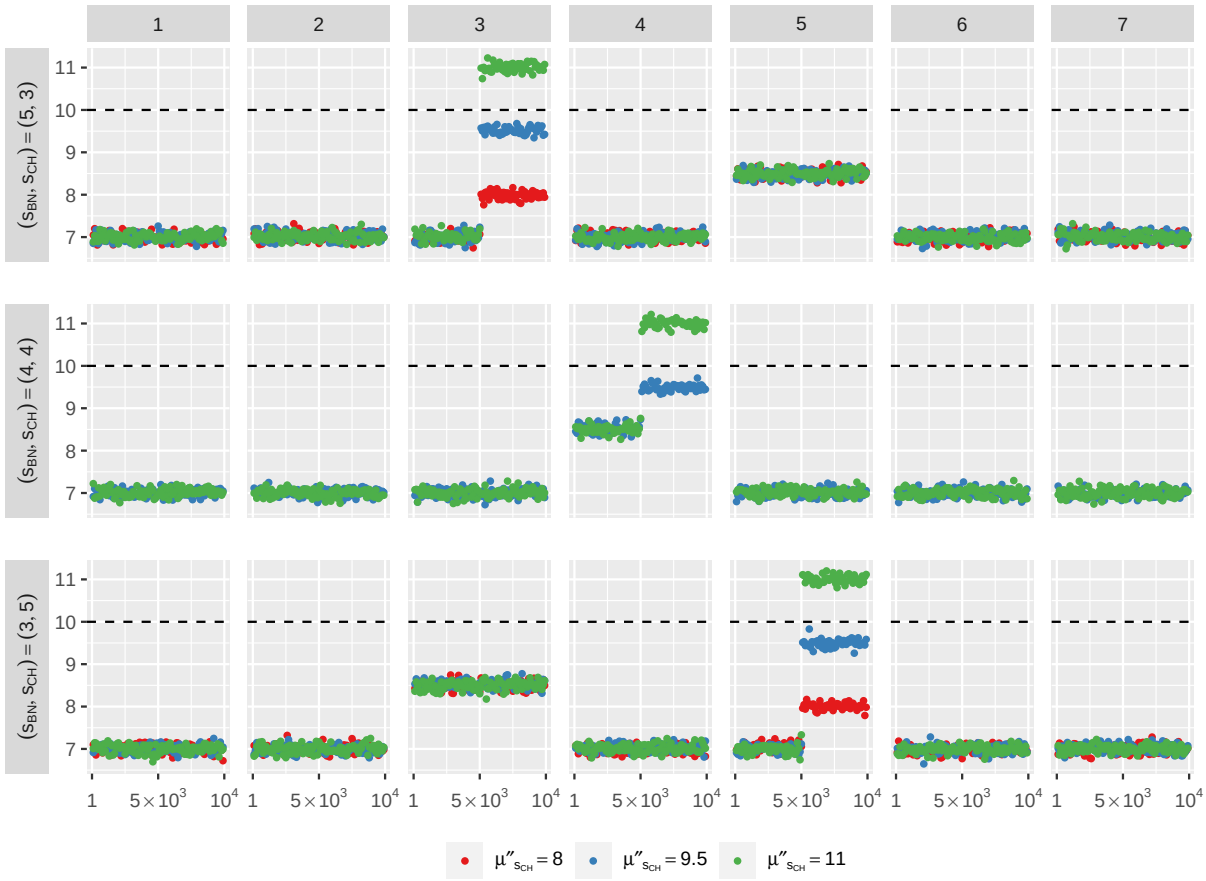


Fig. 6.4 Processing time RW KPI behavior considering three processing time increase levels $\mu''_{s_{CH}}$ and three bottleneck - changing stage configurations (s_{BN}, s_{CH}). Models with $cl_s = 6$.

In figure 6.4 Processing_time_MEAN RW KPI is plotted in three bottleneck-changing stage configurations. For each configuration, different increases of mean processing time $\mu_{s_{CH}}$ are displayed with different colors. The graphs show that average Processing_time KPI is equal to the resource average processing time in each stage. The KPI increases accordingly to processing time variations, independently from the relative bottleneck position with respect to the changing stage.

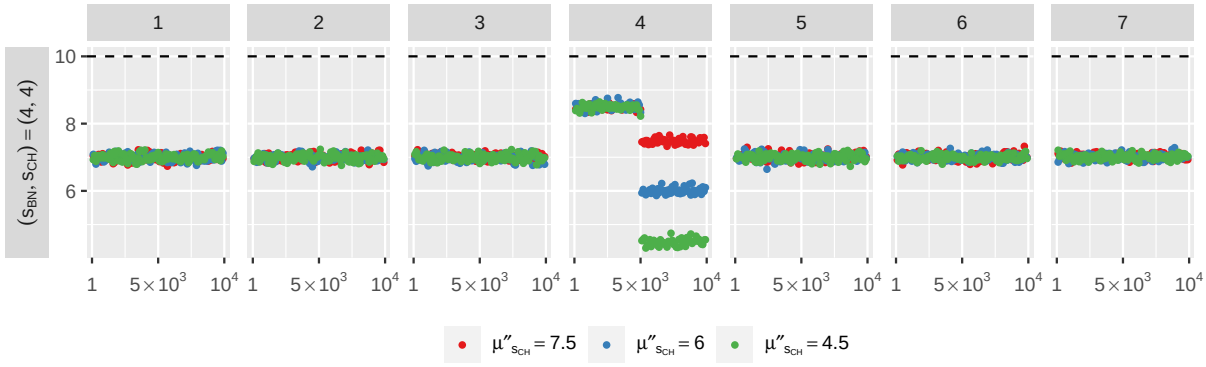


Fig. 6.5 Processing time RW KPI behavior with different processing time decrease levels (models with configuration $(s_{BN}, s_{CH}) = (4,4)$ and $cl_s = 6$)

Plots in figure 6.5 show that the Processing_time presents the same behavior in case of processing time reductions: the KPI is always aligned to processing time in the stage.
It is useful to compare Processing_time_MEAN with Mid_diff_MEAN. Indeed, Processing_time alone tells stage production capacities, but not how much stages work with respect to the line throughput. To obtain this information, these two KPIs are combined calculating Utilization KPI.

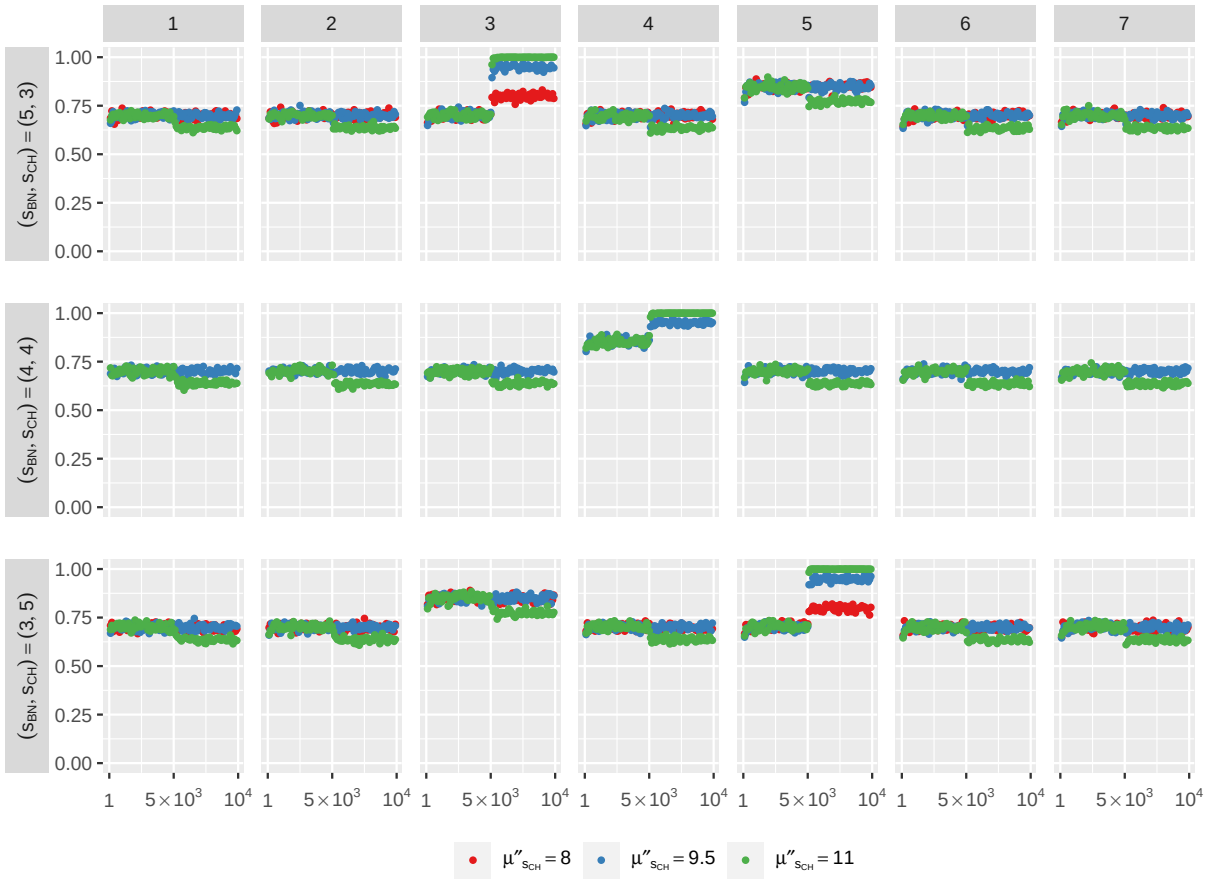


Fig. 6.6 Utilization KPI behavior with different processing time increase levels (models with $cl_s = 6$)

Figure 6.6 portrays Utilization KPI calculated on the same models of figure 6.4. This KPI exhibits some particular features

- Values taken by Utilization are limited to the range $[0, 1]$.
- Like Processing_time, Utilization grows when resource production capacity reduces.
- When the system becomes unstable, Utilization of the bottleneck stage becomes equal to 1, and Utilization of other stages decreases. This is visible in models with $\mu''_{s_{CH}} = 11$.

This behavior is explained considering that this KPI is an indicator of the resource utilization. Indeed, in an unstable system, since cycle time is aligned with bottleneck stage processing time, resource utilization in the bottleneck is equal to 100%. Moreover, other stages of the line have to synchronize their throughput with the bottleneck one, lowering the respective utilization. The

same phenomenon occurs with Utilization KPI: when the system becomes unstable, average Mid_diff becomes equal to average Processing_time of the bottleneck stage (as seen in figure **??**), and so Utilization KPI takes value 1 in correspondence of the bottleneck, and reduces in all other stages.

Processing_time and Utilization KPIs usefulness These KPIs are necessary to get insights about the resources: average Processing_time tells the absolute value of a resource average time needed to process a job. Instead, Utilization shows how much a resource is exploited, comparing its processing time to the cycle time of the line.

6.1.3 Blocking_time, Starving_time and respective Stage State KPIs

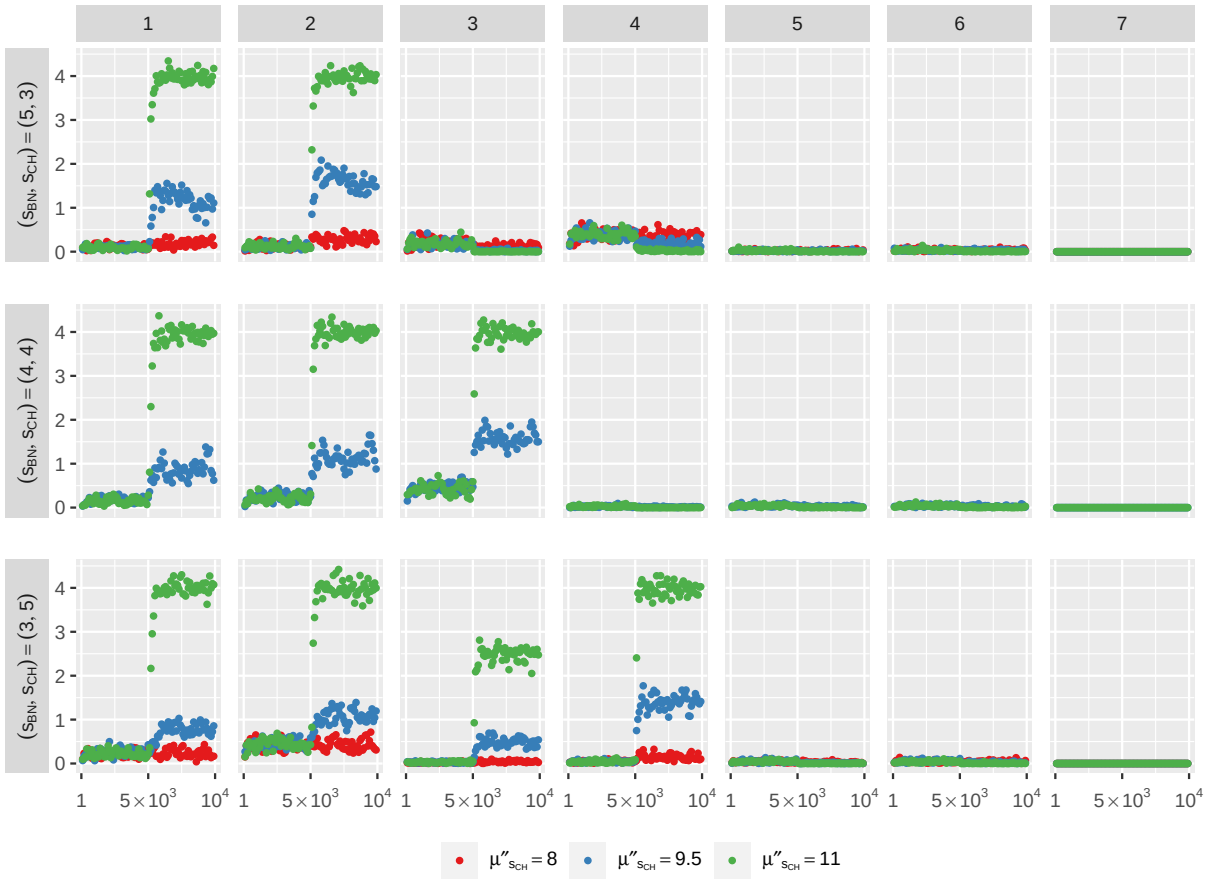


Fig. 6.7 Blocking time RW KPI behavior with different processing time increase levels (models with $cl_s = 6$)

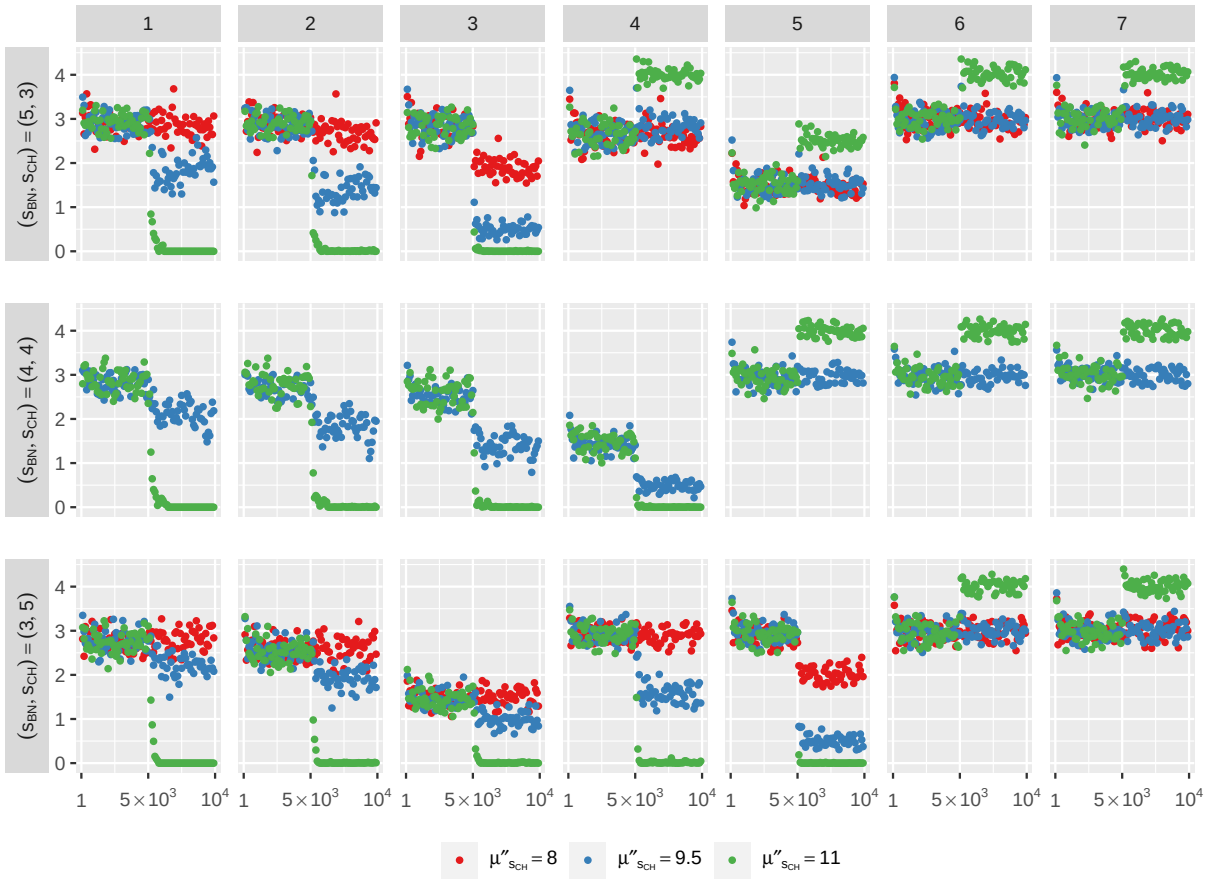


Fig. 6.8 Starving time RW KPI behavior with different processing time increase levels (models with $cl_s = 6$)

These KPIs are discussed together because of their strong relationship: they both depend on processing time and cycle time and their behavior is almost specular. Figure 6.7 and figure 6.8 portray, respectively, Blocking_time_MEAN RW KPI and Starving_time_MEAN RW KPI, plotted on the same models: three bottleneck-changing stage configurations are considered and, for each configuration, different increases of mean processing time $\mu_{s_{CH}}$ are displayed with different colors.

The plots show that, as long as the system is stable, average Blocking_time is greater than 0 in stages upstream the bottleneck, and tends towards 0 downstream and in correspondence of the bottleneck. Instead, Starving_time is greater than 0 in every stage. This is visible in all the models until Case ID reaches value 5000.

Considering a situation where the changing stage s_{CH} looses production capacity, but the system

remains stable, it is possible to summarize Blocking_time and Starving_time behaviors in this way

- Average Blocking_time increases in stages upstream s_{CH}, and reduces in correspondence and downstream s_{CH}. This is visible in figure 6.7, in models with $\mu''_{s_{CH}} = 8$ and $\mu''_{s_{CH}} = 9.5$.
- Average Starving_time decreases in stages upstream and in correspondence with s_{CH}, and increases in stages downstream s_{CH}. This is visible in figure 6.8, in models with $\mu''_{s_{CH}} = 8$ and $\mu''_{s_{CH}} = 9.5$.

It is noteworthy that Blocking_time and Starving_time variations in stage s are less and less relevant the further stage s is from the changing stage s_{CH}, both upstream and downstream. Therefore, plots referring to the changing s_{CH} are the most descriptive and useful to detect and understand the change extent.
It is interesting to observe the KPIs in extreme conditions, which are reached when mean processing time $\mu_{s_{CH}}$ increases to the point that the system becomes unstable. In this situation, average Blocking_time grows and stabilizes around a certain value I_s in stages upstream s_{CH} and drops to 0 in correspondence and downstream s_{CH}; average Starving_time becomes 0 in stages upstream and in correspondence with s_{CH} and increases downstream s_{CH}, reaching and stopping around value I_s. This is visible in models with $\mu''_{s_{CH}} = 11$.
Observing the plots, it can be noticed that I_s is equal to the difference between Mid_diff and Processing_time of a stage. Therefore, I_s is a limit that depends on the cycle time as seen by stage s, and on stage s production capacity.
Given these characteristics, Blocking_time KPI and Starving_time KPI behave as indicators of, respectively, blocking and starvation duration. Indeed, in a stable system, blocking is high in stages upstream the bottleneck, and almost null in stages downstream and in correspondence. Conversely, starvation is lower in stages upstream and in correspondence with the bottleneck, and higher downstream. When a system is unstable, both blocking and starvation have a superior limit: upstream the bottleneck, when buffers become saturated, blocking delays resources release until the bottleneck has finished to process a job; so, blocking lasts, for each resource, the time difference between the processing end in the bottleneck and the processing end in the considered stage. Symmetrically, downstream and in correspondence of the bottleneck, starvation leaves resources idle until the bottleneck has finished to process a job; starvation lasts, like blocking upstream, the time difference between the processing end in the bottleneck and the processing end in the considered stage.
When the system is stable, the limit I_s is still present, and it restrains not Blocking_time or

Starving_time individually, but their sum. Actually, the sum of Blocking_time and Starving_time is always equal to the difference between Mid_diff and Processing_time in a stage because of the KPI definitions (see subsection 4.2.2). It seems reasonable, given the characteristics of the considered systems: since failures or setups never occur, resources can find themselves in three possible states, which are seized and working state (processing), seized and not working state (blocking), and idle state (starvation). The sum of total time spent by a resource in these states gives the whole available time of the resource, and, analogously, Processing_time, Blocking_time, and Starving_time add up matching Mid_diff.
Given that the sum Blocking_time and Starving_time is equal to I_s, the proportions of Blocking_time and Starving_time in the sum depend (also) on the buffer capacity limits of the stages.

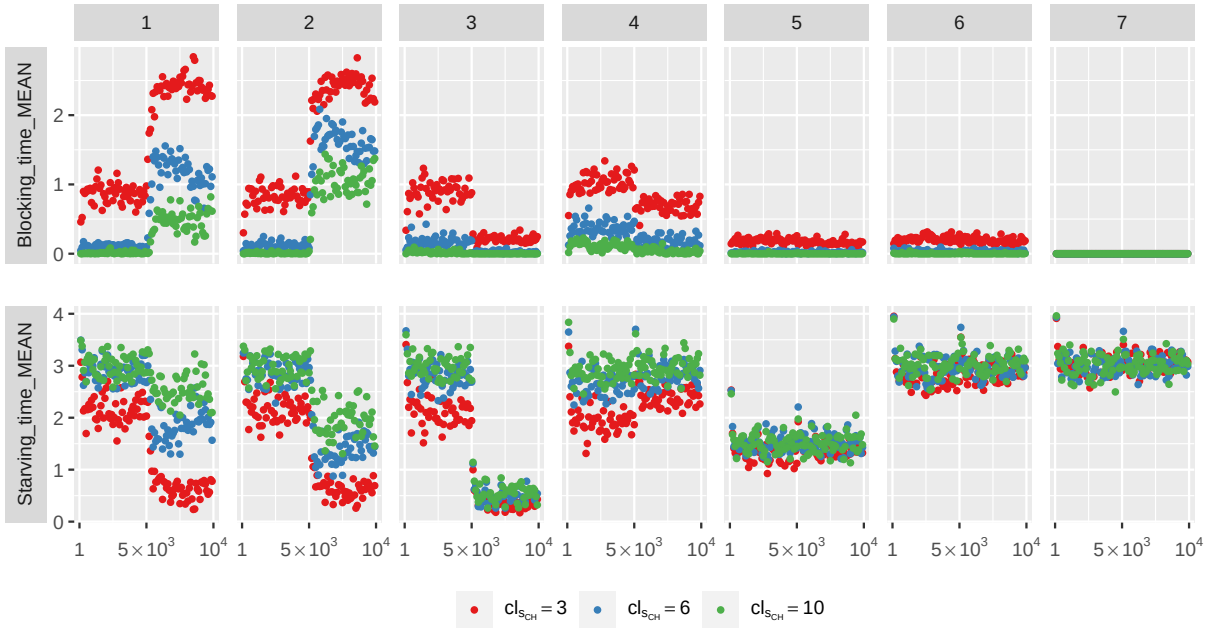


Fig. 6.9 Blocking time and Starving time RW KPIs behavior in case of processing time increase considering different buffer capacity limits (models with configuration $(s_{BN}, s_{CH}) = (5,3)$ and $\mu''_{s_{CH}} = 9.5$)

Looking at figure 6.9, it is possible to conclude that the higher the capacity limit is, the more significant Starving_time becomes in the sum, while Blocking_time contribution lowers. This is particularly visible comparing the model with $cl_s = 3$ and the one with $cl_s = 6$.

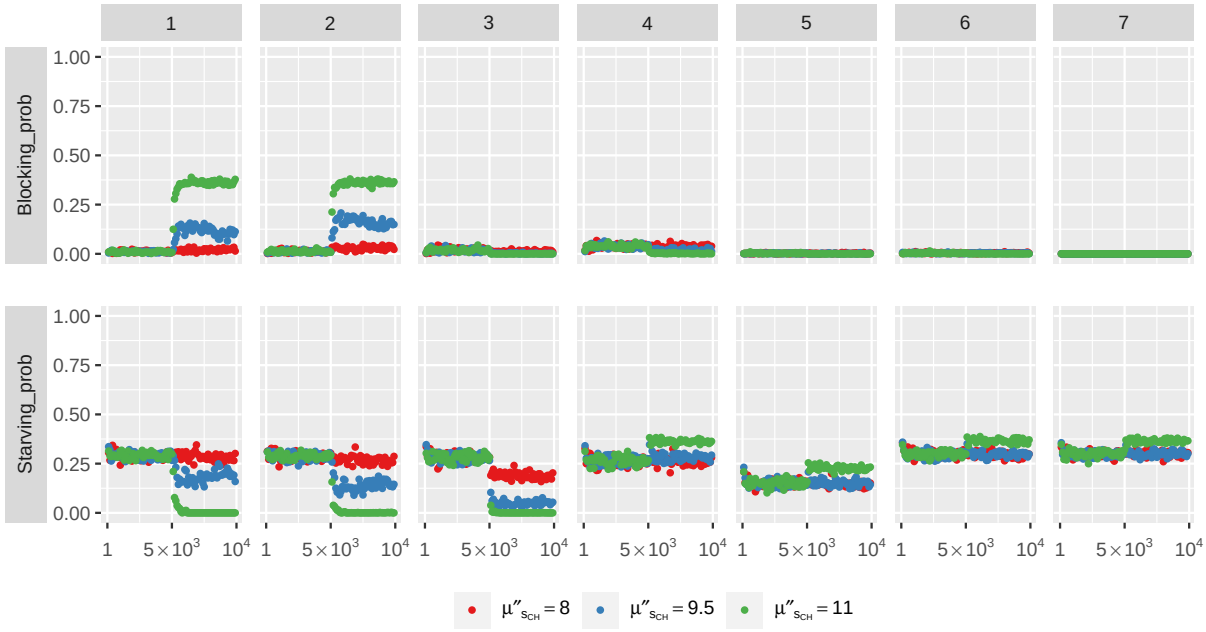


Fig. 6.10 Blocking prob and Starving prob KPIs behavior with different processing time increase levels (models with configuration $(s_{BN}, s_{CH}) = (5,3)$ and $cl_s = 6$)

Like Processing_time with Utilization, also Blocking_time and Starving_time can be represented in combination with Mid_diff. The obtained KPIs are respectively Blocking_prob and Starving_prob: they indicate how much of the available time was spent by the resource without being utilized. Figure 6.10 portrays an example of these KPIs.

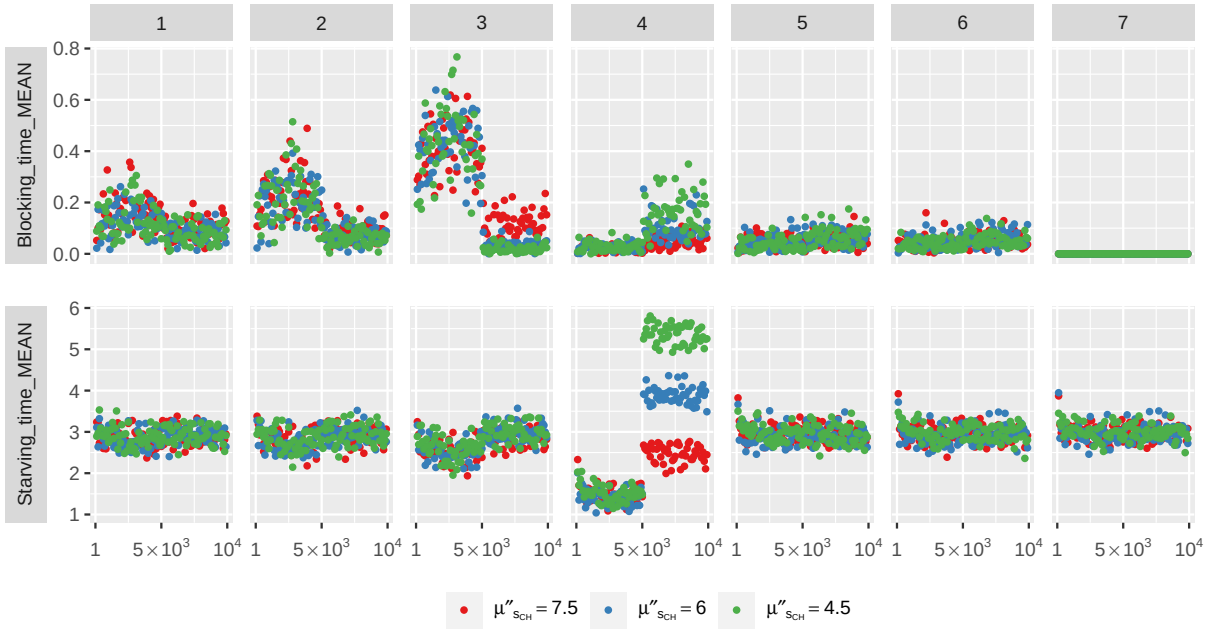


Fig. 6.11 Blocking time and Starving time KPIs behavior with different processing time decrease levels (models with configuration $(s_{BN}, s_{CH}) = (4,4)$ and $cl_s = 6$)

In closing, so far in the discussion about Blocking_time and Starving_time only mean processing time increases have been considered, however the previous observations are similarly valid also in case of processing time reductions. Figure 6.11 displays Blocking_time and Starving_time in this type of variation. In general:

- Blocking_time decreases upstream s_{CH} and increases in correspondence and downstream.
- Starving_time increases upstream and in correspondence with s_{CH} and decreases downstream.

As for processing time increase, also Blocking_time and Starving_time variations caused by processing time reductions are more significant in stages near the changing stage s_{CH}, and become almost irrelevant moving away from s_{CH}.

Blocking_time and Starving_time KPIs usefulness These KPIs allow to monitor processing time variation effects on the stages near the changing one. Indeed, Processing_time focuses only on the changing stage, instead Blocking_time and Starving_time show stage behavior variations along the whole line.

6.1.4 Waiting_time and Average_Queue KPIs

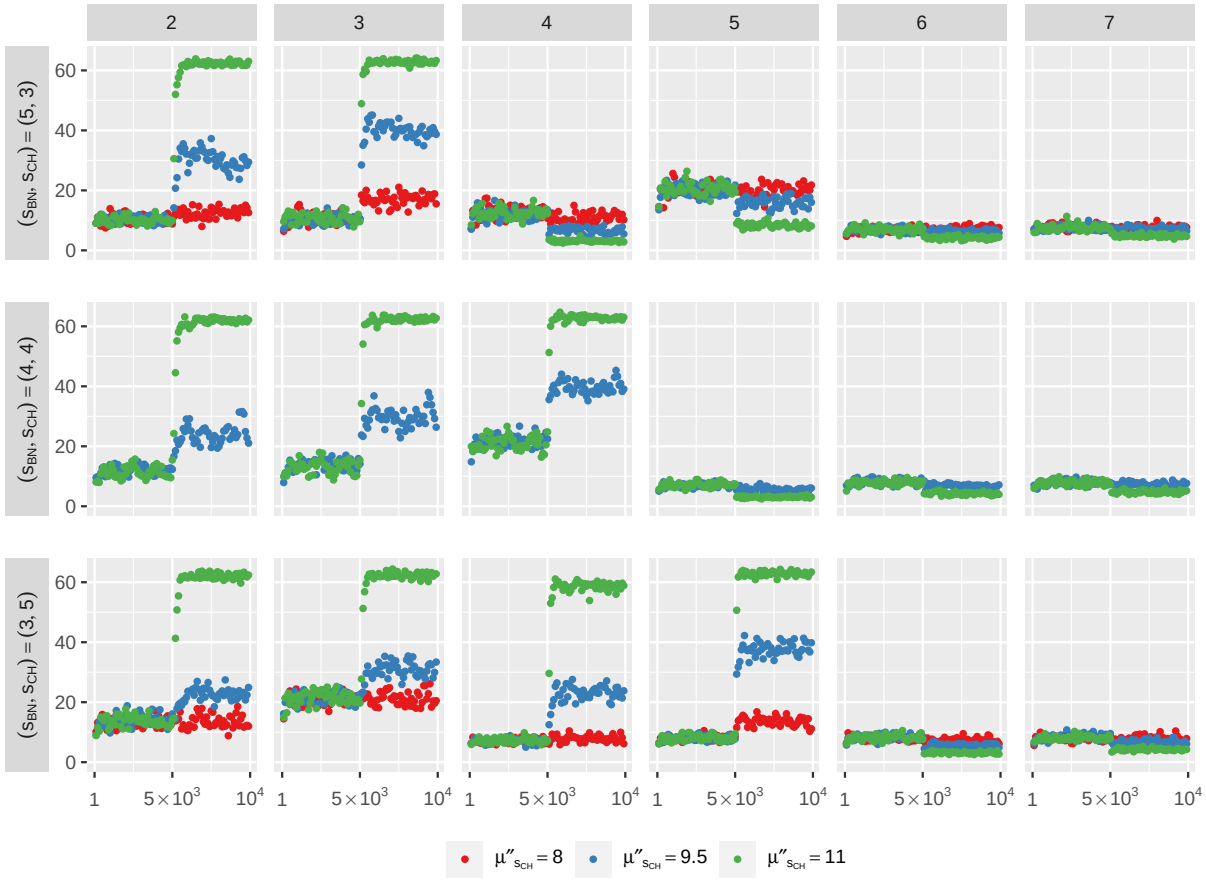


Fig. 6.12 Waiting time RW KPI behavior with different processing time increase levels (models with $cl_s = 6$)

In figure 6.12 Waiting_time_MEAN RW KPI is plotted in three bottleneck-changing stage configurations. For each configuration, different increases of mean processing time $\mu_{s_{CH}}$ are displayed with different colors. The behavior of this KPI can be summarized as follows

- In any moment (i.e. before and after the variation): Waiting_time is higher in stages upstream and in correspondence with the bottleneck, and lower in stages downstream.
- After a processing time growth:
  - Waiting_time increases in stages upstream and in correspondence with the changing stage

– Waiting_time decreases in stages downstream the changing stage

Like with Blocking_time and Starving_time, also a Waiting_time change is more relevant in correspondence of the changing stage s_{CH}, and the processing time variation effects lessen moving away from s_{CH}. Moreover, the plots show that, in each stage, before and after a variation, this KPI stabilizes around a certain value LW_s, which depends on the queue length in stage s buffer: the longer the queue, the higher LW_s is. In an extreme case, when the system becomes unstable, LW_s of stages upstream and in correspondence of the bottleneck is equal to the product of average cycle time (which is equal to the bottleneck processing time) and the stage s buffer maximum job capacity.
Waiting_time combined with Input_diff (which indicates cycle time as seen at the buffer entrances) is utilized to calculate Average_Queue, that is the average number of jobs in a buffer.

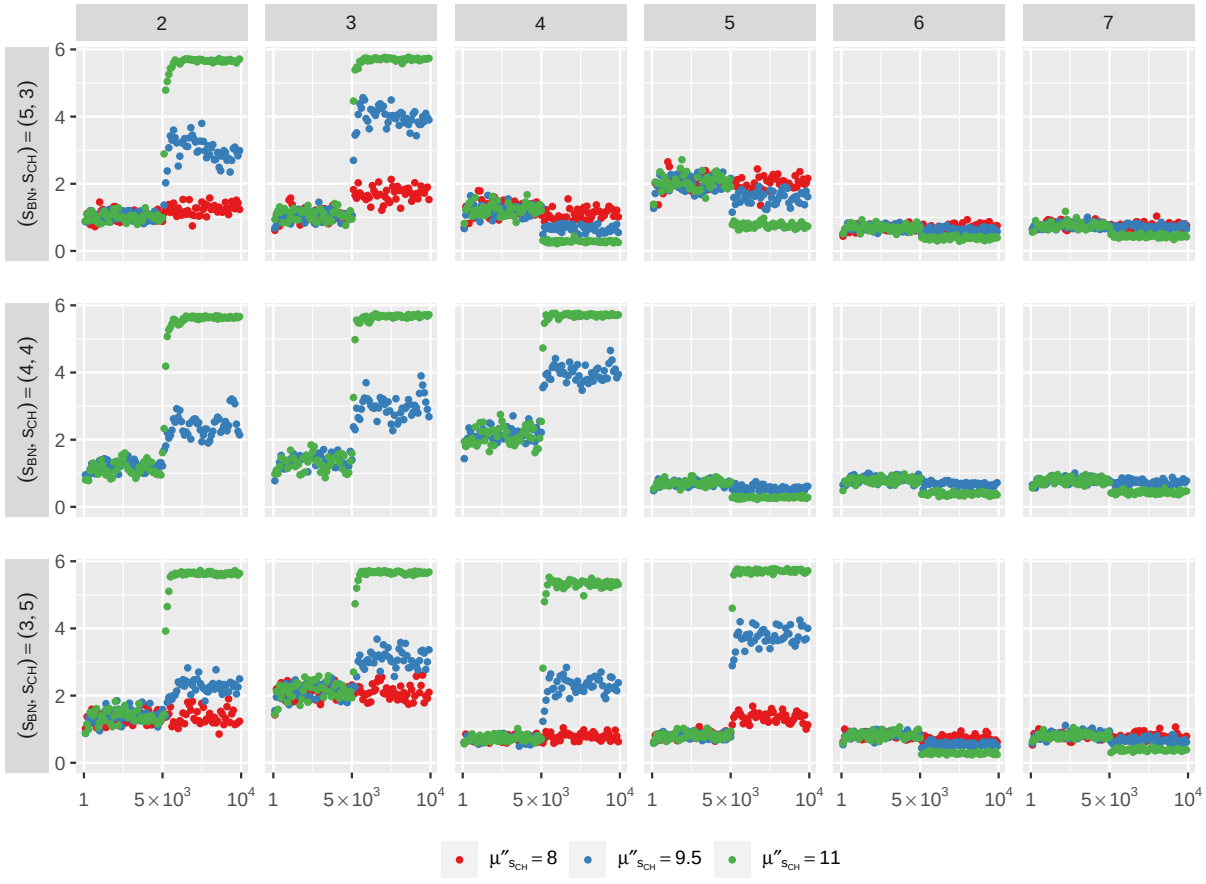


Fig. 6.13 Average Queue KPI behavior with different processing time increase levels (models with $cl_s = 6$)

Figure 6.13 portrays Average_Queue considering the same models of figure 6.12. This KPI behaves alike Waiting_time:

- In any moment (i.e. before and after the variation): Average_Queue is higher in stages upstream and in correspondence with the bottleneck, and lower in stages downstream.
- After a processing time growth:
  - Average_Queue increases in stages upstream and in correspondence with the changing stage
  - Average_Queue decreases in stages downstream the changing stage

When the system becomes unstable, Average_Queue of stages upstream and in correspondence of the bottleneck is equal to the buffer capacity limit. It is to be noted that this is the only situation when it is possible to know with certainty the buffer capacity limits.

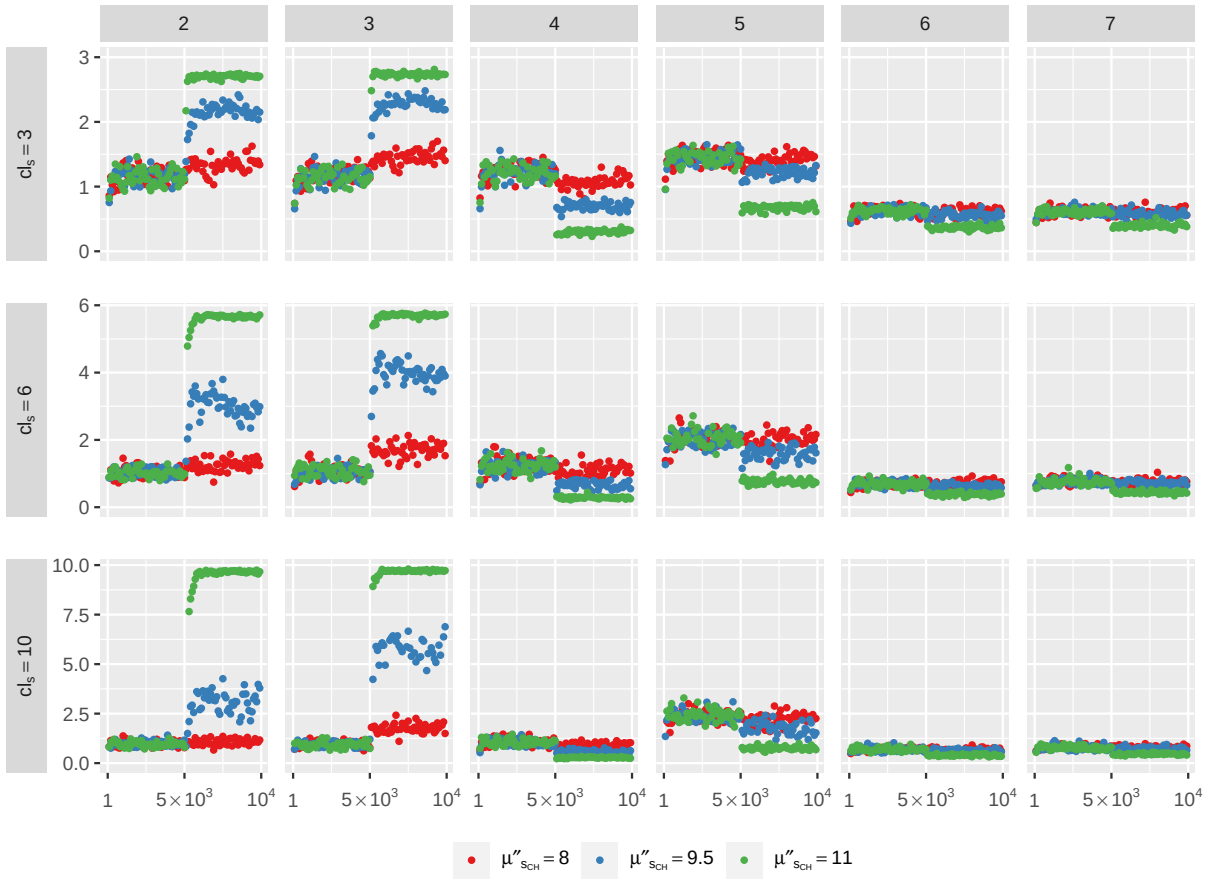


Fig. 6.14 Average Queue KPI behavior with different processing time increase levels considering different buffer capacity limits (models with configuration $(s_{BN}, s_{CH}) = (5,3)$)

Figure 6.14 shows that Average_Queue always increases if the processing time of a downstream stage grows, but it stabilizes immediately below the buffer maximum capacity only when the buffer is completely saturated, that is when the system becomes unstable. This is evident comparing the behavior of Average_Queue in case of $\mu''_{s_{CH}} = 8$ or $\mu''_{s_{CH}} = 9.5$ with its behavior in case of $\mu''_{s_{CH}} = 11$.

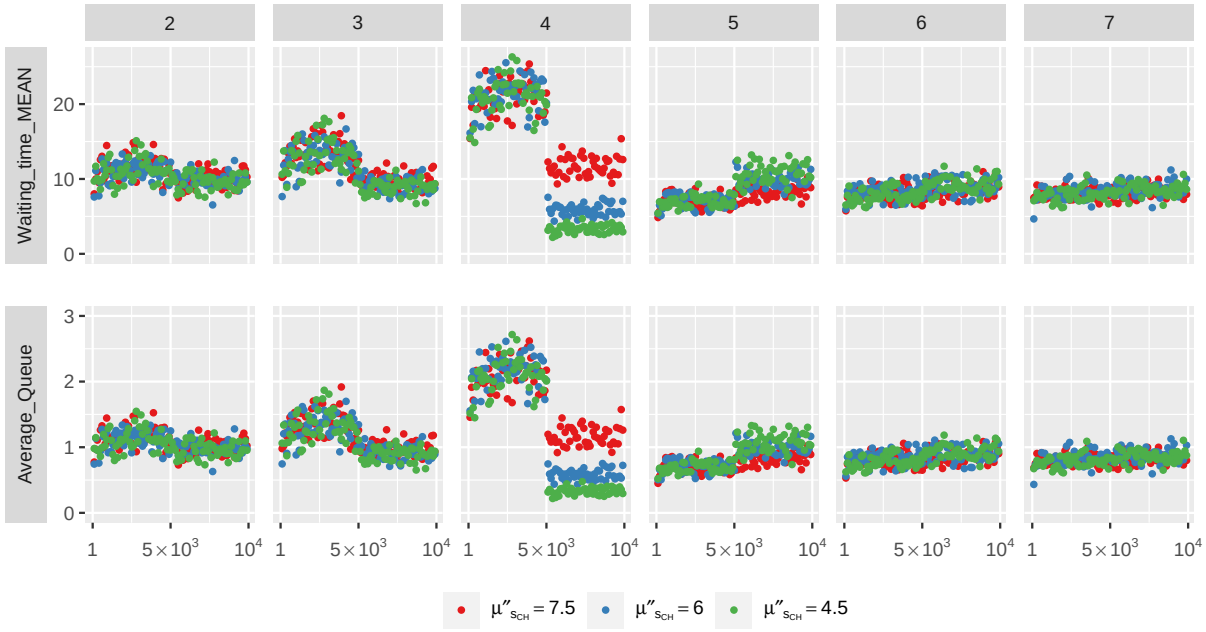


Fig. 6.15 Waiting time RW and Average Queue KPIs behaviors with different processing time decrease levels (models with configuration $(s_{BN}, s_{CH}) = (4,4)$ and $cl_s = 6$)

As visible in figure 6.15, when processing time reduces, Waiting_time and Average_Queue behave the opposite way with respect to the processing time increase case:

- Waiting_time and Average_Queue decrease in stages upstream and in correspondence with the changing stage
- Waiting_time and Average_Queue increase in stages downstream the changing stage

Again, the KPI variations are more relevant the wider is the processing time reduction and the nearer the stage is to the changing one.

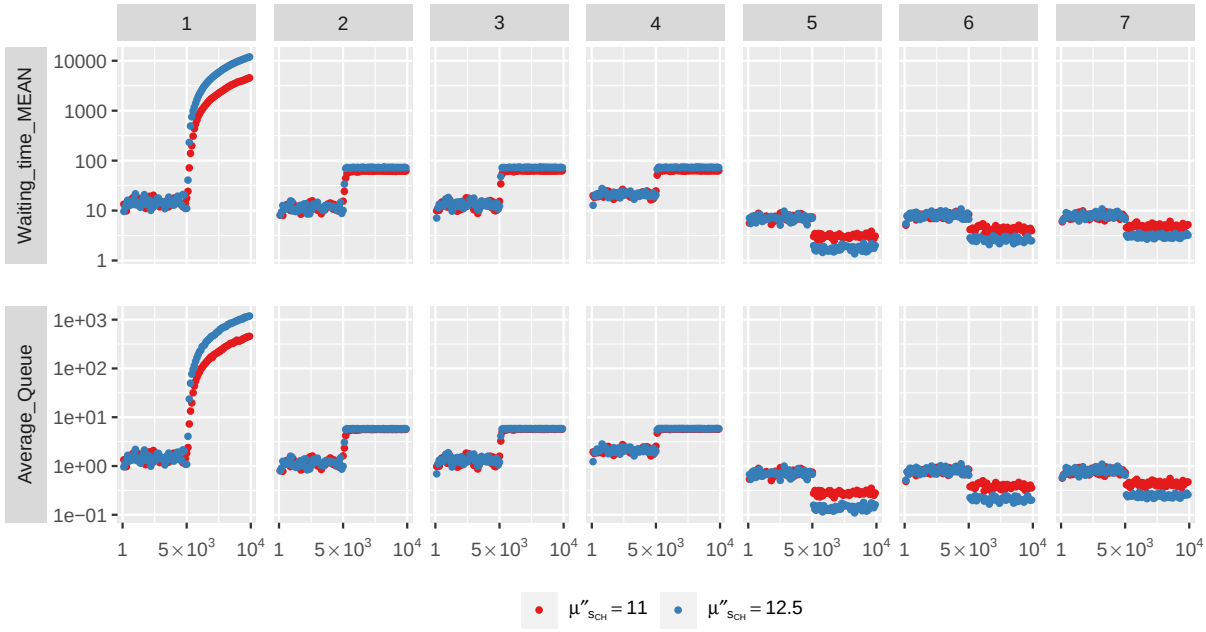


Fig. 6.16 Waiting time RW and Average Queue KPIs behaviors with different processing time increase levels that make the system unstable (models with configuration $(s_{BN}, s_{CH}) = (4,4)$ and $cl_s = 6$)

Lastly, it should be noted that stage 1 was not included in any figure of this chapter until now. As a matter of fact, since the first stage buffer is unlimited, the information of how much time jobs spend in it and how many there are is not useful. Moreover, these KPIs infinitely grow when the system becomes unstable, a circumstance in which Average_Queue even becomes a wrong indicator of the average number of jobs in the buffer: indeed, at the end of the process, no more jobs enter the line, but Average_Queue grows as if new jobs kept arriving, even when the buffer starts emptying. This phenomenon, which is visible in figure 6.16 in plots relative to the first stage, verifies because in this situation Waiting_time increases tending towards infinity, yet Input_diff remains equal to the bottleneck processing time. Reminding how Average_Queue is computed, this means that the numerator keeps growing while the denominator is fixed, making the KPI erroneously continuously increasing. Because of these irregular behaviors, first stage plots have not been displayed in previous figures.

Waiting_time and Average_Queue KPIs usefulness Waiting_time is necessary to have insights about the buffer saturation levels, showing how much time jobs have to wait in buffers. Instead, Average_Queue allows to know how many jobs are on average simultaneously present in a buffer.

6.2 Buffer capacity variation

This type of variation never has effects on resource processing time and, therefore, does not influence the line cycle time. So, a buffer capacity variation is visible only in KPIs related to buffers. This section starts showing and explaining Waiting_time and Average_Queue KPIs.

6.2.1 Waiting_time and Average_Queue KPIs

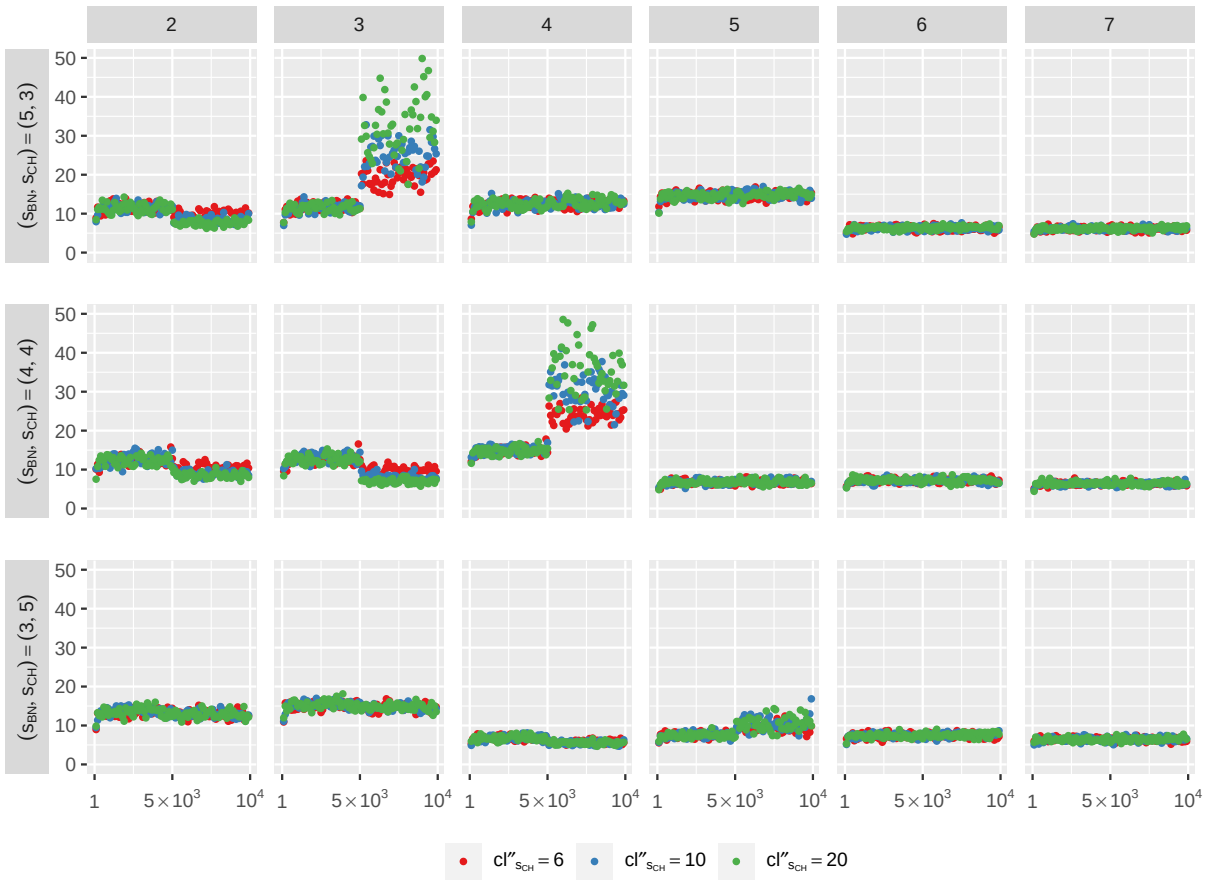


Fig. 6.17 Waiting time RW KPI behavior with different buffer capacity increase levels (models with $cl'_s = 3$)

In figure 6.17 Waiting_time_MEAN RW KPI is plotted considering models with three bottleneck-changing stage configurations (s_{BN}, s_{CH}). For each configuration, different increases of buffer capacity limits $cl''_{s_{CH}}$ are displayed with different colors.
So, in case of buffer capacity increase, Waiting_time behaves as follows:

- Waiting_time increases in correspondence with the changing stage
- Waiting_time decreases upstream the changing stage
- Waiting_time does not change downstream the changing stage

Waiting_time variations are wider the bigger the buffer capacity limit change is. Moreover, in stages near the changing stage the variation is more relevant.
Waiting_time variation extent also greatly depends on the relative position of the changing stage with respect to the bottleneck: the KPI variation is wider when the changing stage is upstream or in correspondence with the bottleneck, and much lower otherwise. This is visible in figure 6.17, comparing models with configuration $(s_{BN}, s_{CH}) = (3,5)$ (change downstream the bottleneck) versus models with configuration $(s_{BN}, s_{CH}) = (4,4)$ (change in correspondence with the bottleneck) or $(s_{BN}, s_{CH}) = (5,3)$ (change upstream the bottleneck).

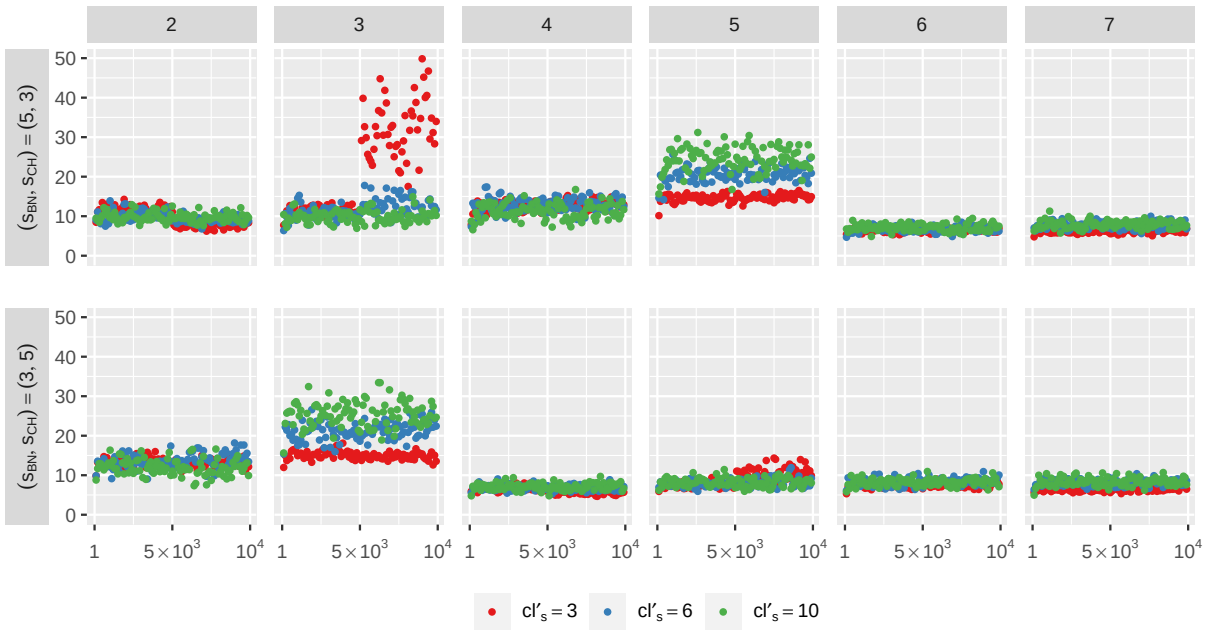


Fig. 6.18 Waiting time RW KPI behavior with different initial buffer capacity levels cl'_s (models with $cl''_s = 20$)

Waiting_time growth also depends on the buffer capacity limit before the change (cl'_s); indeed, the smaller the buffers are, the wider this KPI variation is when a buffer capacity increases. Figure 6.18 portrays that clearly: Waiting_time variations are more significant in models with small and saturated buffers ($cl'_s = 3$) than the ones with bigger and mostly unsaturated buffers

($cl'_s = 6$). Eventually, a buffer capacity limit increase becomes unnoticeable when buffers are extremely unsaturated ($cl'_s = 10$).

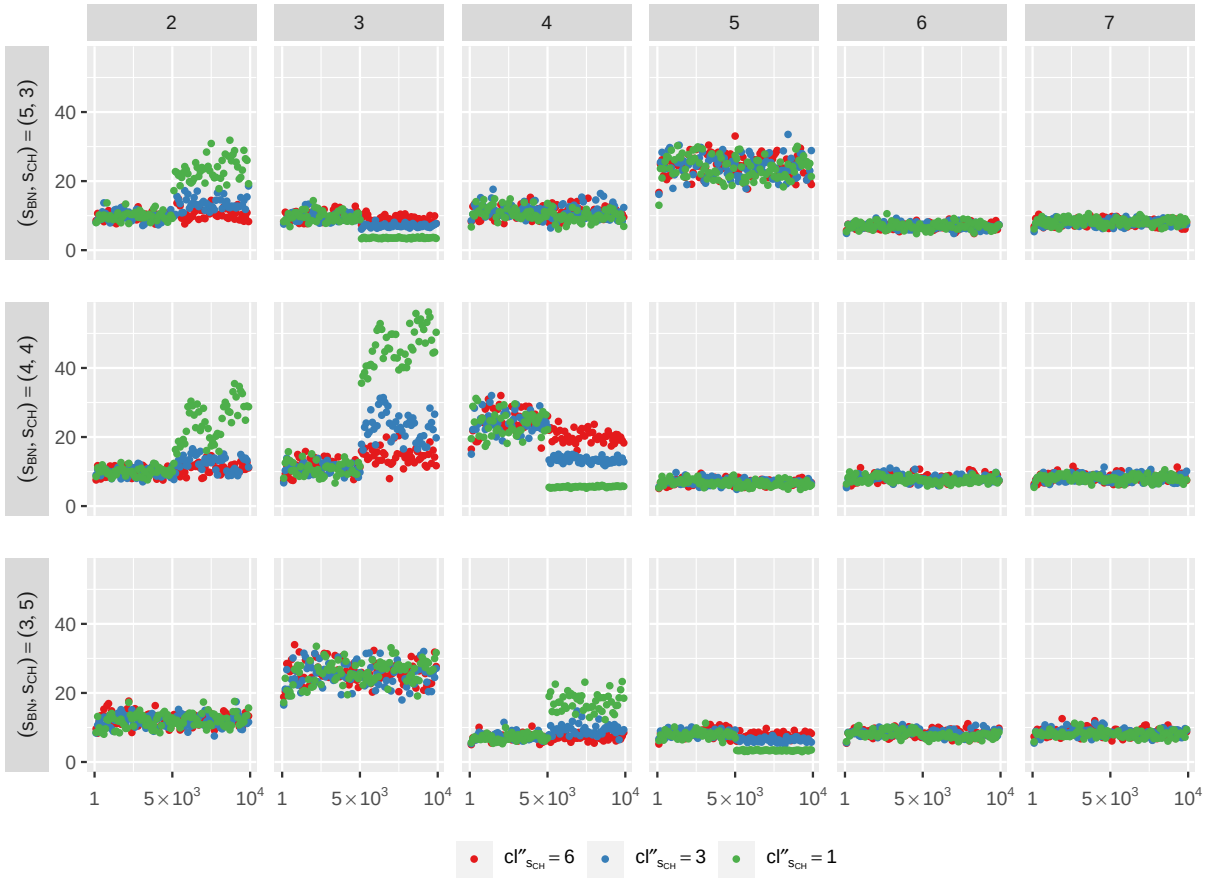


Fig. 6.19 Waiting time RW KPI behavior with different buffer capacity decrease levels (models with $cl'_s = 10$)

When the buffer capacity limit reduces, Waiting_time behaves conversely, as shown in figure 6.19

- Waiting_time decreases in correspondence with the changing stage
- Waiting_time increases upstream the changing stage
- Waiting_time does not change downstream the changing stage

In general, it is important to highlight that not every buffer capacity limit variation is detectable using these KPIs. In situations when, before the change, buffer sizes are much higher than

necessary, and so underused and largely unsaturated during the process, this type of variation is impossible to notice, in particular if it is a capacity increase. Indeed, if buffer queues are consistently much lower than the limit, a variation of the limit will not cause an actual growth of the queue, since the processing time in the previous stage is not changed, and therefore the just added buffer room remains unused.
Average_Queue behaves analogously to Waiting_time.

6.2.2 Blocking_time, Starving_time and respective Stage State KPIs

Fig. 6.20 Blocking time RW KPI behavior with different buffer capacity increase levels (models with $cl'_s = 3$)

Fig. 6.21 Starving time RW KPI behavior with different buffer capacity increase levels (models with $cl'_s = 3$)

Blocking_time and Starving_time are analyzed together, as in previous section.
Figure 6.20 and figure 6.21 portray, respectively, Blocking_time_MEAN RW KPI and Starving_time_MEAN RW KPI, plotted on the same models. Looking at these plots, it is possible to deduce that

- Blocking_time decreases, and Starving_time increases in stages upstream the changing stage
- Blocking_time (slightly) increases, and Starving_time (slightly) decreases in stages downstream and in correspondence with the changing stage.

The variation extent of these KPI depends on different factors

- It is wider the bigger the buffer capacity limit change is.

- It is wider the nearer the stage is to the changing stage.
- It is wider if the changing stage is upstream or in correspondence with the changing stage. This is visible comparing models with configuration $(s_{BN}, s_{CH}) = (3,5)$ (change downstream the bottleneck) versus models with configuration $(s_{BN}, s_{CH}) = (4,4)$ (change in correspondence with the bottleneck) or $(s_{BN}, s_{CH}) = (5,3)$ (change upstream the bottleneck)

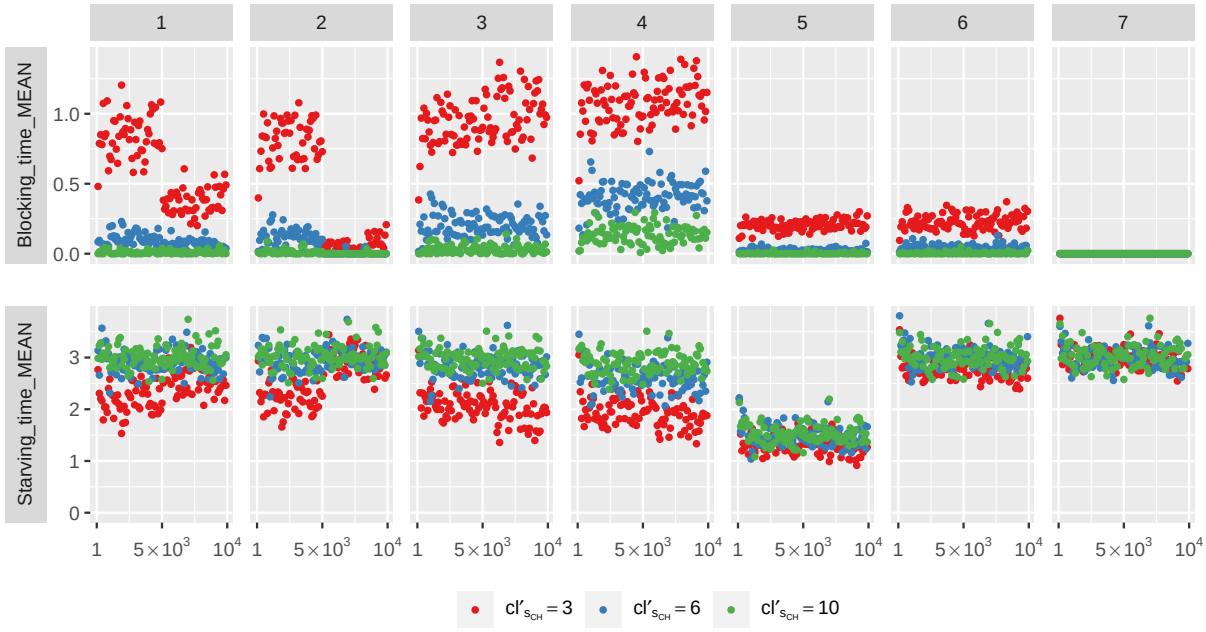


Fig. 6.22 Blocking time and Starving time RW KPIs behaviors with different initial buffer capacity levels cl'_s (models with configuration $(s_{BN}, s_{CH}) = (5,3)$ and after-change value $cl''_s = 20$)

Moreover, variations are more relevant in lines composed of stages with small buffers. This is shown in figure 6.22. When buffers are very spacious (as in models with $cl_s = 10$), changes of both KPIs are not visible at all.

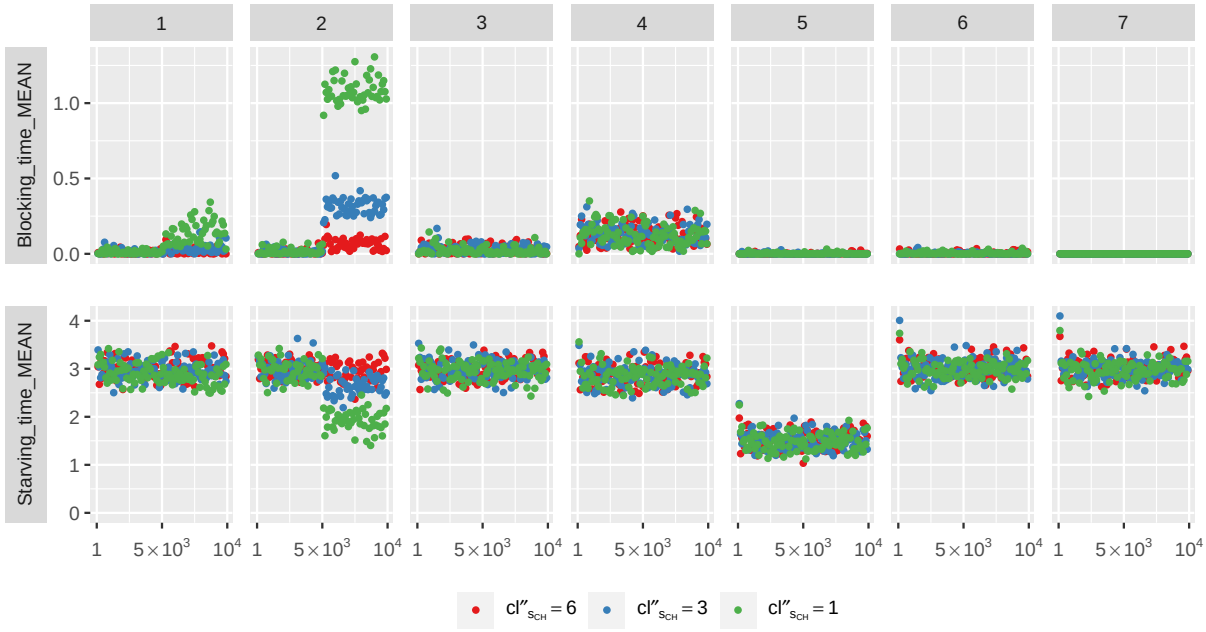


Fig. 6.23 Blocking time and Starving time RW KPIs behaviors with different buffer capacity decrease levels (models with configuration $(s_{BN}, s_{CH}) = (5,3)$ and $cl'_s = 10$)

Considering the case of buffer capacity limit reduction, figure 6.23 portrays Blocking_time_MEAN RW KPI and Starving_time_MEAN RW KPI behaviors

- Blocking_time increases, and Starving_time decreases in stages upstream the changing stage
- Blocking_time (slightly) decreases, and Starving_time (slightly) decreases in stages downstream and in correspondence with the changing stage.

Chapter 7

A map of variations

Chapter 8

Conclusions

References

[1] Leslie Lamport. *Latex*. Addison-Wesley, 1994.

[2] Peter Hertel. Writing articles with latex. 2010.

[3] Simon Fear. Publication quality tables in latex, 2005.

Appendix A

Appendix