Master thesis in Industrial and Management Engineering

Change detection and identification in Simple Serial Lines

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

to my loving parents

thanks for your patience

Abstract

According to the Industry 4.0 paradigm, Digital Twins of production systems must be always aligned with the real systems to guarantee an effective decision making process in a continuously changing environment. To keep the alignment, digital process models can be updated with Process Mining techniques through data collected by sensors. This thesis proposes a model-update procedure and addresses the issue of detecting and identifying changes occurring in Simple Serial Lines, a particular type of production lines. Using event logs, which are data structures generated by line embedded sensors, KPIs are computed, plotted, and analyzed to get insights of the system behavior. Simulation is used to test the effectiveness of the procedure.

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

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the last three centuries humankind experienced three different industrial revolutions that deeply modified every aspect of society, not only changing production and economic models, but also generating new cultural movements and wide political unrests across the 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 the 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, AT&T Bell Laboratories invented the transistor. 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, generating a growth of productivity, efficiency and quality in manufacturing. Processor capability development and diffusion in every field of the human activities have gone along with data storage capacity increase, which in turn 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 [2]. 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 [3], implying that these technological developments are considered so significant that it is now widely acknowledged we are entering in the Fourth Industrial Revolution.

1.1 Industry 4.0 components

What does the Fourth Industrial Revolution consist of?

To understand the complex landscape of Industry 4.0, I will summarize the definitions of the main facets of this new manufacturing model, then I will outline how these concepts are linked to each other.

Cyber-Physical System (CPS) A CPS is defined in [4] 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 [4], 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*" [5]. 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 system: it imitates and changes in accordance with the real system. However, this relationship is not one-directional, having the DT that, in turn, while running in parallel, influences the real system, allowing to foresee the process evolution through simulation and to design improvements.

Big Data Analytics (BDA) Big Data, as written in the HACE theorem [6], "*starts with large-volume, heterogeneous, autonomous sources with distributed and decentralized control, and seeks to explore complex and evolving relationships among data*". The data characteristics stated by the definition 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 analysis techniques to extract meaningful information, such as Data Mining and Process Mining.

Cloud Computing (CC) 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.

Although these concepts do not have well-defined borders, and even partially overlap, figure 1.1 was included to suggest a logical connection among them. The main aspect that jumps out from the previous definitions is that IoT and DT are two sides of the same coin, which is CPS: IoT can be considered the hardware side and DT the software side of a CPS. The CPS monitoring and the coordination of its parts are performed using BDA techniques, which are hosted on digital environments capable of storing and analyzing huge amount of data (i.e. CC).
Ultimately, information extraction and exchange are the glue that brings all the Industry 4.0 facets together. Indeed, the main innovative side that characterize the Fourth Industrial Revolution is

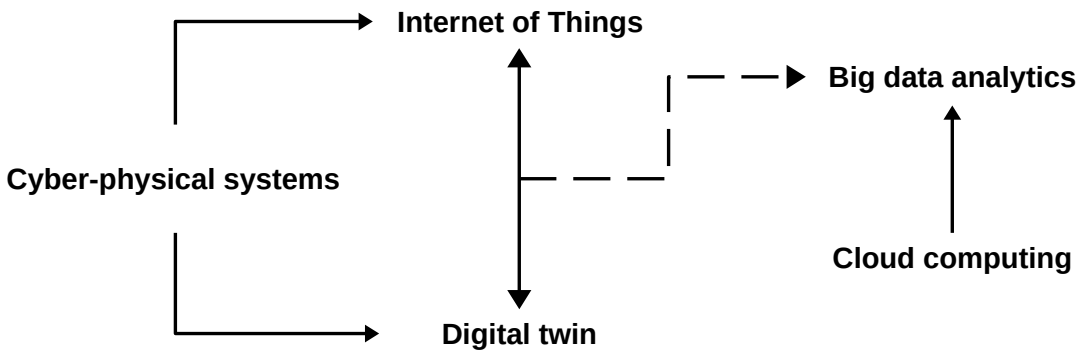


Fig. 1.1 Industry 4.0 scheme

the exploitation of huge amount of data, aiming to automatically improve efficiency and quality in manufacturing contexts.

1.2 Tools of Big Data Analytics

BDA works as a bridge between the hardware side and the software side of a CPS, extracting information from the data collected by sensors embedded in the real system (IoT) and from data obtained by simulations run on the digital copy (DT). In general, the activity of obtaining information from big data volumes is called Knowledge Discovery in Databases (KDD), defined in [1] as "*the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data*". KDD does not belong to a specific field of application, indeed it is a framework that can also adopted in manufacturing contexts. As figure 1.2 shows, KDD includes many steps:

1. Understanding the manufacturing domain: preliminary research of the manufacturing sector which the data come from, and specification of the analysis goals
2. Collection of the data: raw data gathering from different sources
3. Data cleaning, pre-processing and transformation: data pre-processing to remove noise, replace missing values and redundancies, and selection of a dataset structure appropriate for data mining algorithms application
4. Data integration: integration of data from heterogeneous sources

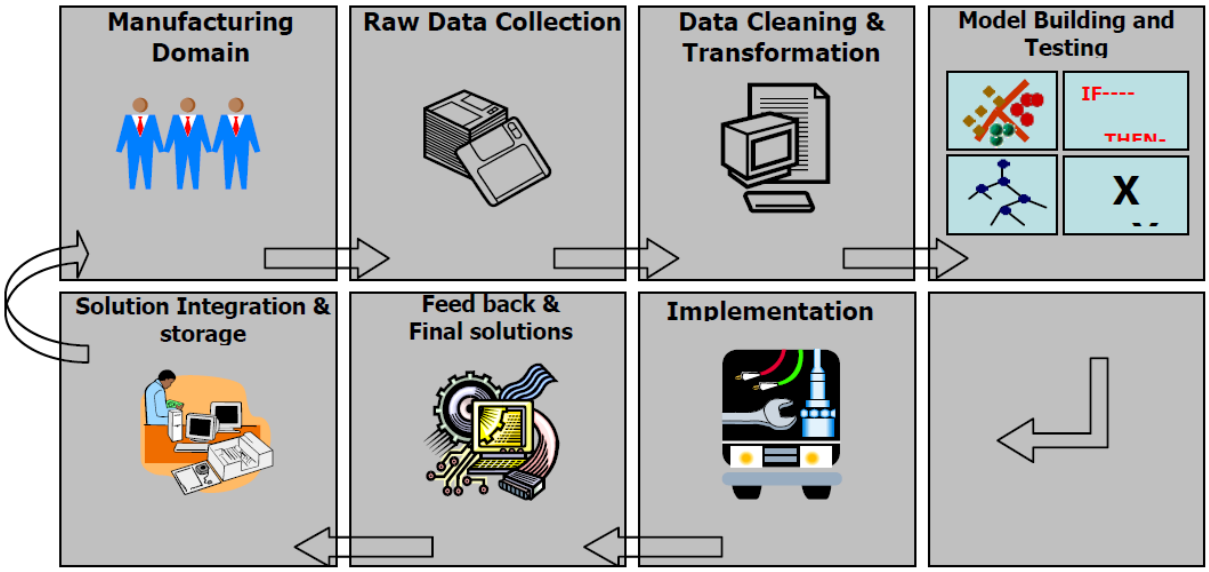


Fig. 1.2 KDD in manufacturing [1]

5. Selection of Data Mining area (see 1.2.1): selection of the Data Mining technique to use for the data analysis
6. Selection of the appropriate data mining algorithm: selection of the Data Mining algorithms to use for the data analysis
7. Interpretation and Visualization: study of the results and preparation of graphs to support the conclusions
8. Implementation of discovered knowledge: execution of the required system modifications and feedback collection
9. Knowledge storage, reuse and integration into manufacturing system: storage of the acquired know-how in sight of future reuse

1.2.1 Data Mining

The core of KDD is Data Mining (DM), that is the usage of specific algorithms for identifying patterns in data to uncover hidden relationships (Descriptive DM) and foresee outcomes (Predictive DM). It is possible to distinguish some DM main macro-areas:

- **Characterization & Discrimination**: this DM branch aims to summarize data to provide a concise and information-rich description of the analyzed system.

It is widely used in manufacturing contexts to address quality control problems or to get a overall sight of a process.

- **Classification**: this DM branch aims to map (i.e. classify) the data into one of many different predefined classes.
  In manufacturing contexts, it is particularly used to classify defects and to perform online control chart pattern recognition.

- **Clustering**: this DM area branch aims to map the data into one of many different not-predefined classes; these classes are called cluster, characterized by being natural grouping of data items based on similarity metrics.
  Some examples of manufacturing usage are its application to study supply chain and order picking routines, and to design part families and machine cells in cellular manufacturing.

- **Prediction**: this DM area branch aims to build and use models to foresee the class of unlabelled samples, or to assess the value or value range of an attribute that a given sample is likely to have.
  It is widely used in manufacturing contexts, for example to design efficient resource maintenance policies, predicting the deterioration of components and machines.

- **Association**: this DM branch aims aims to identify relationships patterns among items in a database, not based on inherent properties of the data, but rather are based on frequency of data-items co-occurrences.
  An example of its application in a manufacturing context is the usage of association to identify an efficient disposition of items stocked in a warehouse.

These DM macro-areas are deeply interwined and different approaches can be concurrently applied to explore data and extract meaningful information.

1.2.2 Process Mining

In recent years, data analysis research brought to the design of highly specialized and efficient DM algorithms, devised for specific areas of application. This leaning towards specialization lead to the definition of different DM fields, whose developments are separately pursued, even if mutual influences are still present. Some DM field examples are Text Mining, that aims to extract information from written sources, and Image Mining, that focuses on studying hidden

patterns in pictures and photos. Among the other, in the late 1990s, Process Mining has been introduced.

Process Mining (PM) develops across multiple disciplines beyond DM, involving also on Artificial Intelligence research and business process modeling. PM has the purpose of "*discover, monitor and improve real processes by extracting knowledge from event logs readily available in today's (information) systems*" [7], where event logs are data structures that register the events occurred during a process execution (see section 3.1).

As the definition suggests, there are three main types of PM [8]:

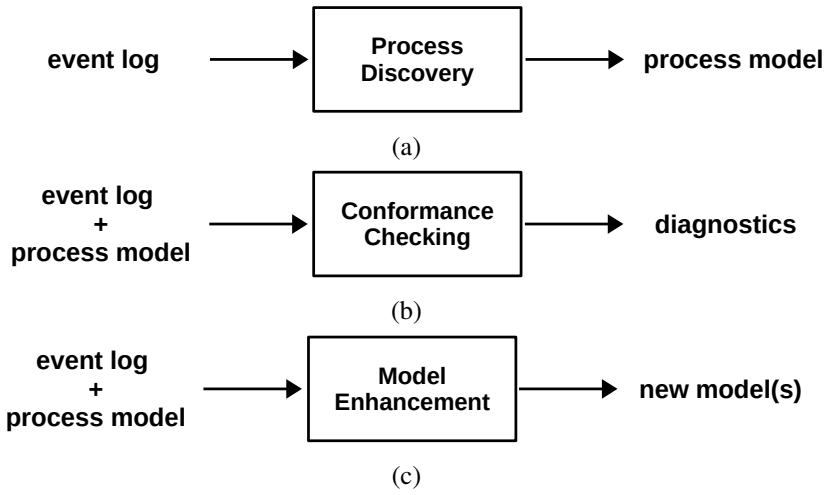


Fig. 1.3 The three main Process Mining branches, highlighting input and output of each one

- **Process Discovery**
  As figure 1.3a shows, this PM branch focuses on extracting information from the event log in order to understand the logical dependencies among the activities constituting the process. In other words, it aims to build a process model (i.e. a model describing the system behavior) looking at what activities are performed on cases (i.e. jobs) passing through the system.

- **Conformance Checking**
  As figure 1.3b shows, this PM branch focuses on the comparison of the behavior of cases registered in an event log with the process behavior prescribed in an already existing process model, looking for mismatches between the actual system (i.e. the one recorded by the event log) and theoretical system (the one represented in the model).

- **Model Enhancement**
  As figure 1.3c shows, this PM branch focuses on the enrichment of an already existing

process model through the analysis of an event log related to the same system. In particular, the process is studied from different points-of-view, such as the resource perspective (e.g. Organizational Mining, which gives insights about the social network structure of a system) and the case perspective (e.g. time an sequence analysis, that allows to easily build Gantt Charts).

PM has been extensively applied in business and healthcare issues [9], and recent researches also tried to apply it to the manufacturing context. Indeed, event logs, which are the initial data source for any PM algorithm, are obtainable as outputs of sensors embedded in production lines, making PM a promising candidate as useful tool for CPS implementation (PM can be adapted to help in the DT construction) and maintenance. In particular, concerning the second part, it is critical to design efficient methods to keep DTs aligned with the real systems, and this thesis aims to suggest a new approach and to take the first steps into a new model updating process, employing the PM framework as a starting point.

1.3 Thesis structure

In the second chapter the thesis scope is defined, exploring the state of the art and stating the dissertation goals. The third chapter describes the study boundaries, specifying what type of manufacturing systems, sensors, and process variations are taken into account. The fourth chapter focuses on the indicators used to monitor the process, explaining how they are extracted from event logs. The fifth and sixth chapter concern the experimental tests, respectively describing the simulated system characteristics and analyzing the indicator behaviors in the simulations. The seventh chapter summarizes the results shown in the previous chapter.

Chapter 2

Scope of the work

A recently published article by Lugaresi et al. [10] showed that a process model, 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 explained in section 1.1, a Digital Twin should be a dynamic copy of a real manufacturing system. It means that, ideally, the DT is changed as soon as a variation occurs in the real process, in order to keep the digital copy continuously aligned with the real system.
This thesis has the purpose of laying the foundation for an always up-to-date DT, making it an auto-updating copy of the real system given an initial process model.

2.1 A standardized approach for the process model updates

The model updating procedure, in case a variation occurs in the real system, can be divided in the three following steps.

- **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 process control (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 detect 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. Figure 2.1 schematically represents a Change Detection example.

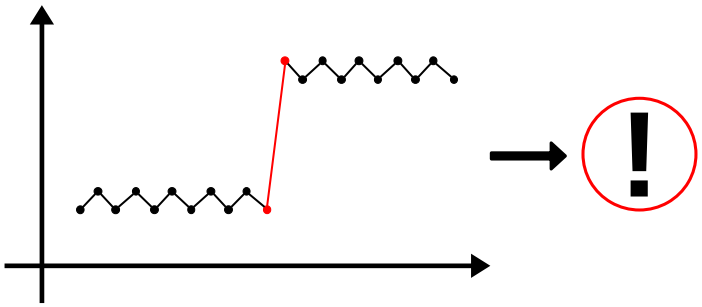


Fig. 2.1 Change Detection

- **Change identification**
  A detected change of one or more indicators is classified to determine what kind of variation occurred in the underlying process. To do so, a "change map" is interrogated, in order to match the indicator detected change with a real system change, and then choose the correct update to the process model. Figure 2.2 schematically represents a Change Identification example.

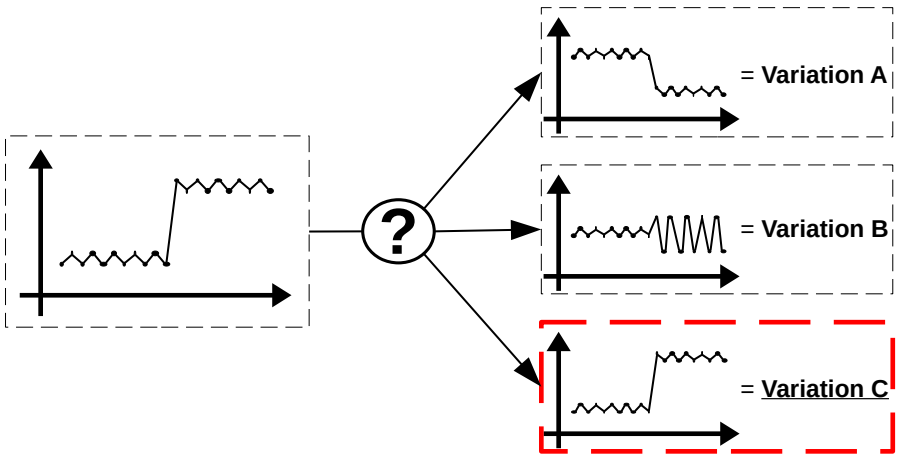


Fig. 2.2 Change Identification

- **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 more focused rather than a general change in the model. Figure 2.3 schematically represents a Model Modification example.

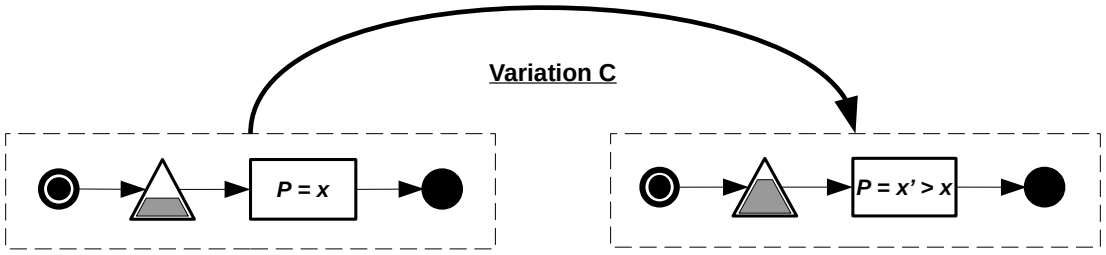


Fig. 2.3 Model Modification

2.1.1 Aim of the work

This 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 whether a change can be detected, not on how to detect it.
At the same time, the thesis 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 regard, the observed indicators are extracted only from event logs generated by line sensors. The reason 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 other sensors and the analysis of different data structures.
To show the novelty of this approach, the following sections are dedicated to a state of the art overview concerning the PM manufacturing applications and the PM adaptations to allow dynamic process model updates in general-purpose applications.

2.2 Process Mining applications in manufacturing

PM has been applied in manufacturing mostly as a process monitoring tool. For example, in [11] PM is utilized to build the model of a production planning process of a manufacturing company using event logs obtained from an ERP system. A similar application of PM is performed in [12], where PM algorithms are adapted to analyze manufacturing make-to-order production processes, focusing on the resource workloads and on the delays caused by issues during the process. It is to be noted that, even if these studies are conducted in manufacturing contexts, they do not actually examine production lines, rather they use PM to analyze the manufacturing systems on a production planning level, and therefore are quite far from the PM application this thesis aims to achieve.
Lugaresi et al. in [10], as previously outlined, instead direct their attention on designing a PM algorithm which can be applied on production lines, in order to easily and swiftly build a DT imitating a real system. The mining procedure presented in the article consists of four steps:

1. Dataset loading: an event log is pre-filtered and pre-processed to generate the data structures suitable for the information extraction.
2. Optimized mining of event types: this is the core of the procedure, consisting in the application to data of the PM algorithm adapted to production line applications in order to obtain an *activity network*, which shows how activities are connected in the system.
3. Petri Net model generation: the *activity network* is translated in an initial Petri Net model[1].
4. Petri Net model adjustment: the initial Petri Net Model is polished to obtain a formally correct Petri Net model representing the real system.

Therefore, the article shows that it is possible to mine the necessary information from a system event log to generate a process model, which can serve as a DT logical base. This thesis inserts on the same line of research, aiming to develop around this method and suggesting a procedure to move from a DT one-off generation to an always up-to-date DT.

2.3 Auto-update of process models using Process Mining

In real applications, event logs often do not present as complete datasets, instead they are generated continuously appending new events recorded in real-time. In other words, an event

[1] Petri Nets are process modelling language

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 PM algorithms are not able to deal with, since they require datasets to be finite. The PM branch that aims to address this problem is called Streaming Process Discovery (SPD), and different solutions have been developed, such as the usage of smarter forms of sampling, and summarizing the data into event frequency tables.
The main issue to confront when dealing with event stream is the occurrence of variations in the real system during the process. In [13] three algorithms are proposed, aiming to build a process model from an event stream and keeping it synchronized with the system. These algorithms are based on the *Heuristic Miner*, one of the most used control-flow discovery algorithms. In [14] a similar technique is applied, introducing a growing and pruning mechanism in order to align the *prefix-tree* representing the activity relationships with the real process. A different approach is taken by [15], where a mixed technique is designed using probabilistic neural nets and an evolutionary algorithm to identify and keep updated a process model with a real manufacturing system.
The model updating processes designed by these articles are much different from the one this thesis suggests. Indeed, in these methods models are influenced by every single event occurring in the system, without an actual monitoring of the process; instead, the approach we propose is based on the detection (and identification) of variations to trigger the model updates. Here the thesis originality lies: before model modifications we think it is critical to understand if and what kind of change is verifying in the real process, in order to minimize the updating computational effort and to be able to build a DT on more stable and reliable system models.

Chapter 3

Thesis boundaries

Production systems can take a wide variety of forms, depending on the specific type of application. Generally, the proper way to explore how to address a problem using a new approach is to start from relatively simple contexts, delineating the dissertation boundaries.
This chapter defines the thesis areas of interest, concerning the type of production systems, the sensor positions and the types of variations assumed to occur during the process. Before the boundary definitions, a description of the event logs is provided, in order to illustrate their structure and to show what information they can carry.

3.1 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 [16].
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**: the code that uniquely identifies a certain case flowing in the process.
- **ID of the activity**: the 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.
However, in general, event logs are composed by more fields containing different case attributes or event attributes, which allow a deeper and more meaningful analysis of the process.

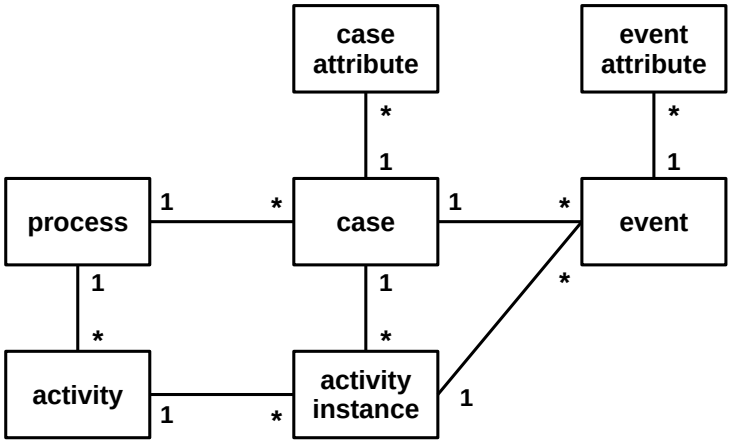


Fig. 3.1 Event log class diagram [8]

Figure 3.1 displays the class diagram of the components of a general event log. The main structure of simple event logs is still present in richer event logs:

- Each event log refers to only one process
- Each case belongs to one process but a process may consist of many cases
- Each event refers to one case but a case may be the subject of many events
- Each activity may be performed on many cases and each case may be processed by different activities

The diagram shows that, in general event logs, in contrast to simple event logs, cases and events may have many attributes. Some examples are

- **Timestamp**: an event attribute containing the instants when events verified
- **ID of the resource**: an event attribute containing the codes that identify which resources (e.g. an operator that performs different activities, a machine in a multi-machine stage) were utilized to process the job

- **Cost**: an event attribute containing the costs related to events
- **Case type**: a case attribute registering the category to which the case belongs, useful in a multi-product system context

The presence of attributes allows to apply Model Enhancement algorithms, which aim to create models that look at a process from different perspectives. For example, if the resource ID[1] is available, it is possible to generate graphs of the social network, that show the employee relationships, the work sequence and the connection frequency.

3.1.1 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.
As explained in section 2.3, in real applications, often, events are collected one at a time, generating event streams, however 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 the aim to clarify if it is possible to detect and identify some specific types of variation. Concerning the change detection, since this issue is addressed by assessing the event log with an event-by-event (or window-by-window) approach, that is comparing each 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

[1] In this case, intended as the operator IDs

stream situations, even if the study only considers complete event logs, the change identification presents additional challenges that are not addressed in this thesis.

3.2 Considered systems

3.2.1 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.

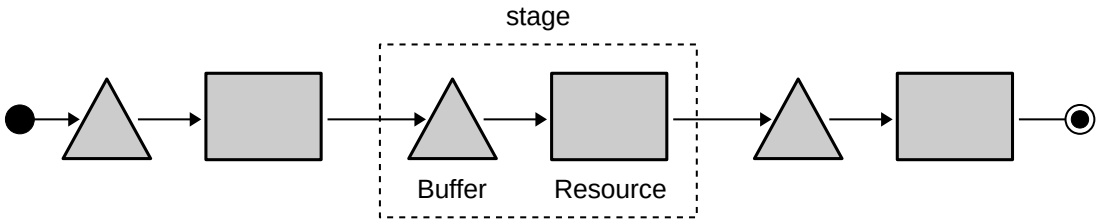


Fig. 3.2 A SSL composed of three stages

Figure 3.2 shows an example of a SSL made of three stages, each one composed of a buffer and a resource. SSLs are described in [17], 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.

- **Single type of product**
  Multi-product systems work many types of jobs, and the job type may change during the process. Job types are distinguished, for instance, by production time or cost or because of

the stage sequence that they have to pass through. SSLs process a single type of jobs, and the type does not change throughout the process.

- **Production and transfer batches composed of one unit**
  Resources does not process groups (i.e. batches) of jobs, but only single jobs. Moreover, jobs move from a stage to the following one alone.

- **FIFO (First In First Out) service policy**
  FIFO 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.
  It is noteworthy that, 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.

3.2.2 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 inter-arrival 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.
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.

3.3 Considered sensor positions

To build a full functioning DT many types of sensors can be involved. For example, in [18] 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 into 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 section 3.2, 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.

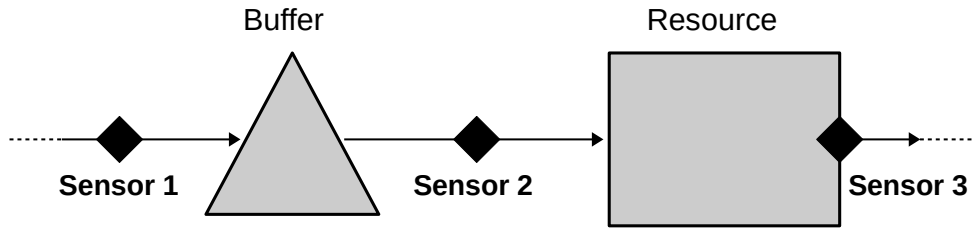


Fig. 3.3 Three sensors configuration scheme

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.

3.4 Considered process variations

3.4.1 Variation classifications

The issue of process variation has been addressed in Process Mining, in particular within the wider topic of Streaming Process Discovery (see section 2.3), that is the construction of models

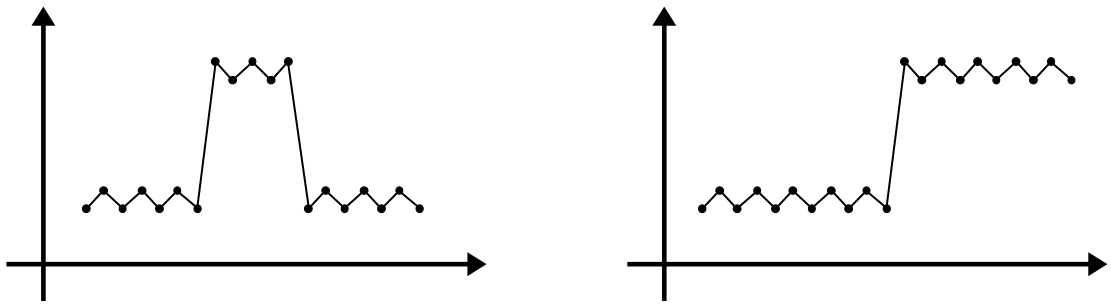
based not on finite event logs but on streams of events. Indeed, in the long term, an event stream relative to a process monitored in real time could be susceptible to concept drifts, which are "*situations where the process is changing while being analyzed*" [8].
In [19] by Bose et al., concept drifts 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, classification on duration 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 3.4a)
- **Permanent changes**: variations lifespans are long-lasting, to the point that can be considered permanent. These changes usually affect all the process activities. (Figure 3.4b)

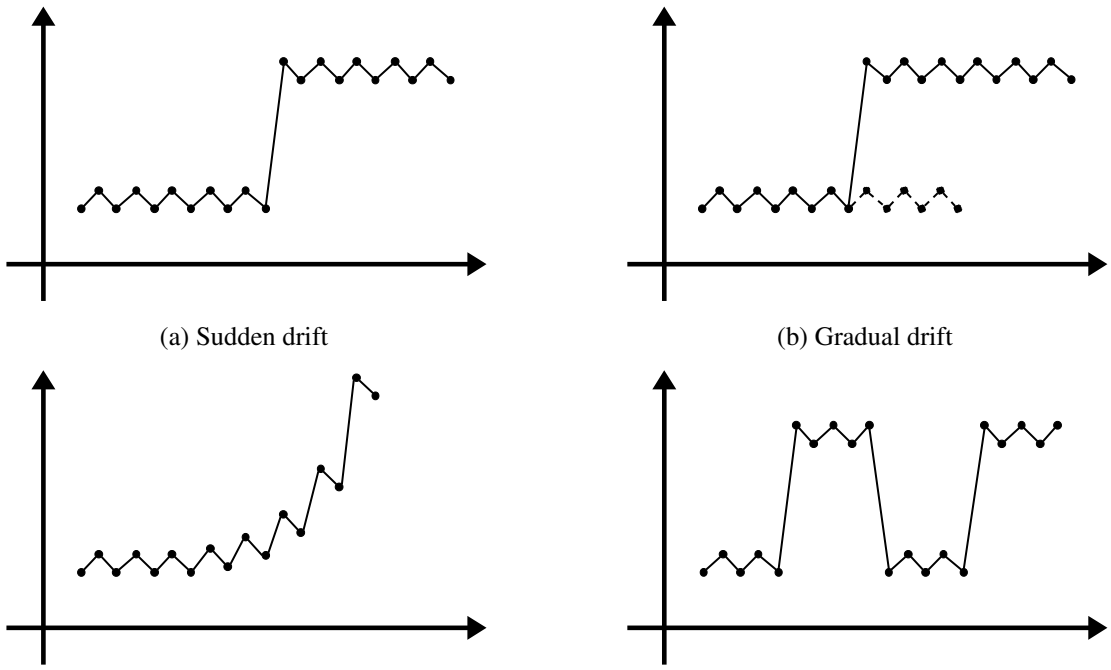


(a) Momentary change (b) Permanent change

Fig. 3.4 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 3.5a)
- **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 3.5b)
- **Incremental drift**: it includes variations that occur through small and incremental steps. This is typical of processes with slow machine deterioration over time. (Figure 3.5c)
- **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 3.5d)



(a) Sudden drift

(b) Gradual drift

(c) Incremental drift

(d) Recurring drift

Fig. 3.5 Change classification on nature

3.4.2 Considered variations

In this thesis two 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 expand 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 4

Indicator extractions from event logs

The identification and extraction of meaningful process indicators (i.e. Key Process Indicators, KPIs), capable of conveying useful information in a simple, concise, and not misleading way, is a critical challenge in Operations and Process Management. In this chapter, a series of KPIs are proposed as candidates to monitor a manufacturing system: the first section describes what fields are assumed to compose event logs generated by production lines, while the second section explains how event logs are transformed in *case lists*, which are data structures more suitable for the suggested indicator extractions. The last section shows how these KPIs are computed starting from *case lists* and provides a brief description of what they represent.

4.1 Assumptions on event logs generated by production lines

Event logs, as described in section 3.1, have a well-defined general-purpose structure, characterized by the association of one row to each event, accompanied by a case identifier and an activity identifier, and also by a variable number of attributes.
In the particular manufacturing application considered by this thesis, it is possible to assume that the sensor produced event logs are 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 4.2, 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 section 3.2, 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 section 3.1: 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 (see section 3.3).

- **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.

Table 4.1 Example of event log generated by sensors embedded in a production line

Event ID	Case ID	Activity ID	Position ID	Timestamp
157480	7500	1	1	106387.78
157481	7500	1	2	106389.37
157482	7500	1	3	106392.58
157483	7500	2	1	106392.58
157484	7500	2	2	106399.19
157485	7500	2	3	106413.20
157486	7500	3	1	106413.20
157487	7500	3	2	106413.20
157488	7500	3	3	106416.90
157489	7500	4	1	106416.90
157490	7500	4	2	106416.90
157491	7500	4	3	106423.33
157492	7500	5	1	106423.33
157493	7500	5	2	106486.05
157494	7500	5	3	106498.81
157495	7500	6	1	106498.81
157496	7500	6	2	106498.81
157497	7500	6	3	106506.34
157498	7500	7	1	106506.34
157499	7500	7	2	106506.34
157500	7500	7	3	106513.06

Table 4.1 is provided to show an example of the described field structure, reproducing an event log portion limited to the trace of a job having CaseID equal to 7500.

4.2 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 datasets obtained modifying event logs are named *case lists*. This conversion is applied to simplify the successive calculations and to organize the data in a structure that better fits the stage-focused study performed through the KPI analysis.
A preliminary transformation is the discharge of the Event ID field. 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, while 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. The Timestamp field is spread in three fields, separating time instants acquired by sensors located in different positions (recorded in Position ID) in the same stage. The resulting fields are named as follows

- **Timestamp_Buffer** (T_B): it registers the instant when the job entered in the stage buffer, recorded in event log rows having Position ID equal to 1
- **Timestamp_Resource** (T_R): it registers the instant when the job entered in the stage resource, recorded in event log rows having Position ID equal to 2
- **Timestamp_End** (T_E): it registers the instant when the resource finished the job processing, recorded in event log rows having Position ID equal to 3

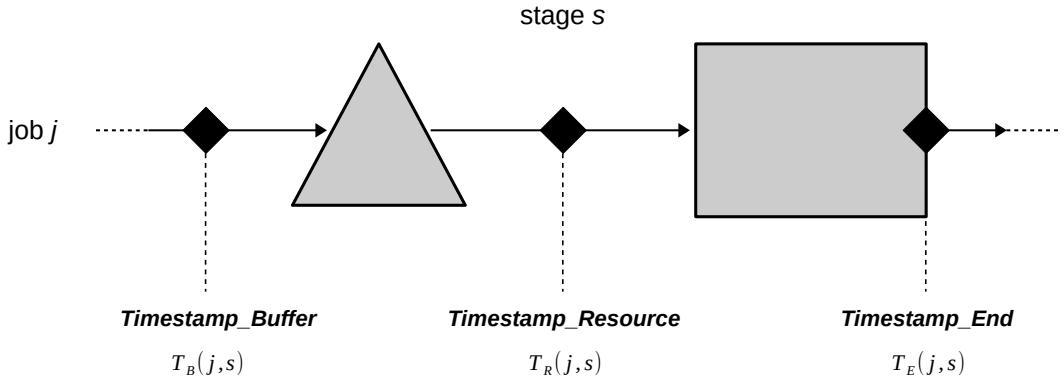


Fig. 4.1 Relations between sensors and timestamps

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.
For the rest of the thesis, Timestamp fields are going to be referenced as follows

- $T_B(j,s)$: 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

Table 4.2 Case list example

Case ID	Activity ID	Timestamp_Buffer	Timestamp_Resource	Timestamp_End
7500.00	1	106387.78	106389.37	106392.58
7500.00	2	106392.58	106399.19	106413.20
7500.00	3	106413.20	106413.20	106416.90
7500.00	4	106416.90	106416.90	106423.33
7500.00	5	106423.33	106486.05	106498.81
7500.00	6	106498.81	106498.81	106506.34
7500.00	7	106506.34	106506.34	106513.06

Table 4.2 is provided to show a case list example, computed starting from the event log portion of table 4.1.

4.3 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.

4.3.1 Basic KPIs

These indicators are computed as differences between timestamps, so they are all time intervals (see figure 4.2). 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.

Same case intervals (SCI) KPIs

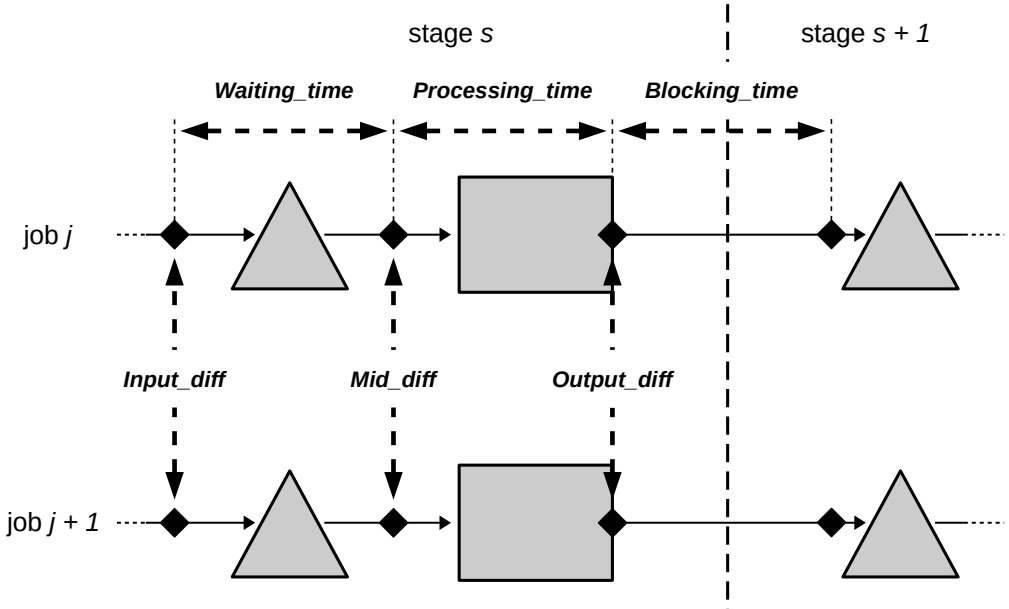


Fig. 4.2 Basic KPI computations scheme

Waiting_time is a SCI indicator, computed as the difference between Timestamp_RESOURCE and Timestamp_BUFFER (see figure 4.3). 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

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

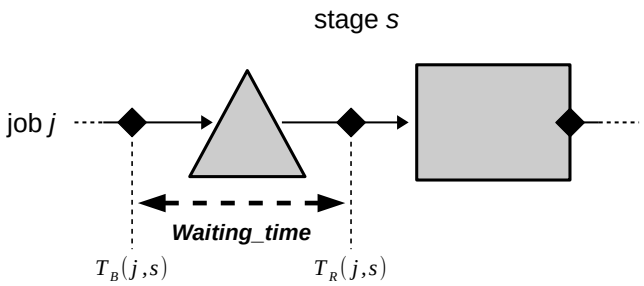


Fig. 4.3 Waiting time computation scheme

Processing_time is a SCI indicator, computed as the difference between Timestamp_END and Timestamp_RESOURCE (see figure 4.4). 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

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

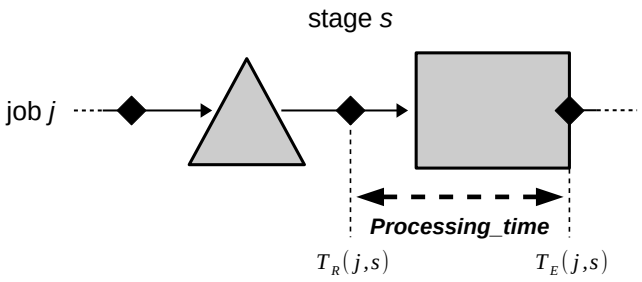


Fig. 4.4 Processing time computation scheme

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 (see figure 4.5). 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

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

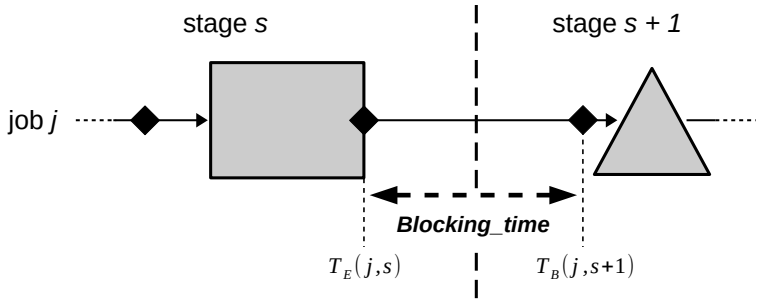


Fig. 4.5 Blocking time computation scheme

Consecutive cases intervals (CCI) KPIs

Input_diff, Mid_diff, and Output_diff are CCI indicators, computed as the differences between pairs of timestamps [1] relative to consecutive job passages through the same sensor (see figure 4.6). 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

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

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

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

[1] Input_diff is related to pairs of Timestamp_BUFFER, Mid_diff is related to pairs of Timestamp_RESOURCE, and Output_diff is related to pairs of Timestamp_END

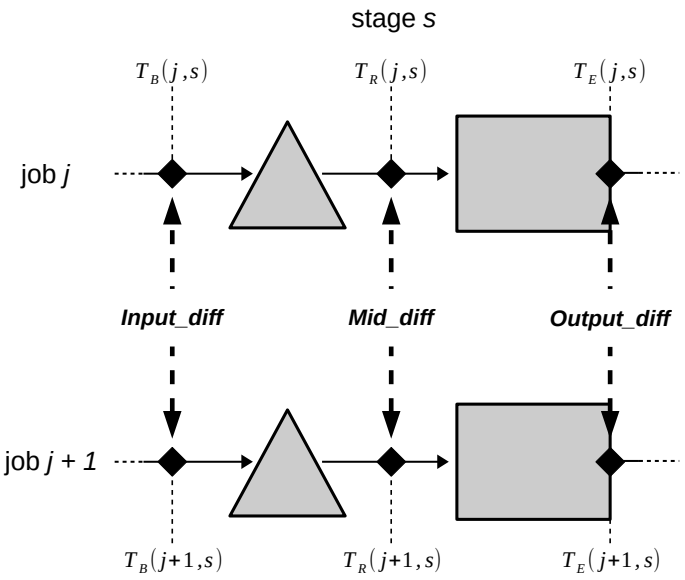


Fig. 4.6 Consecutive cases intervals computations scheme

4.3.2 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 KPI

Starving_time is calculated as the difference between Mid_diff and the sum of Processing_time and Blocking_time (see figure 4.7). 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: first, 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. Second, 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

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

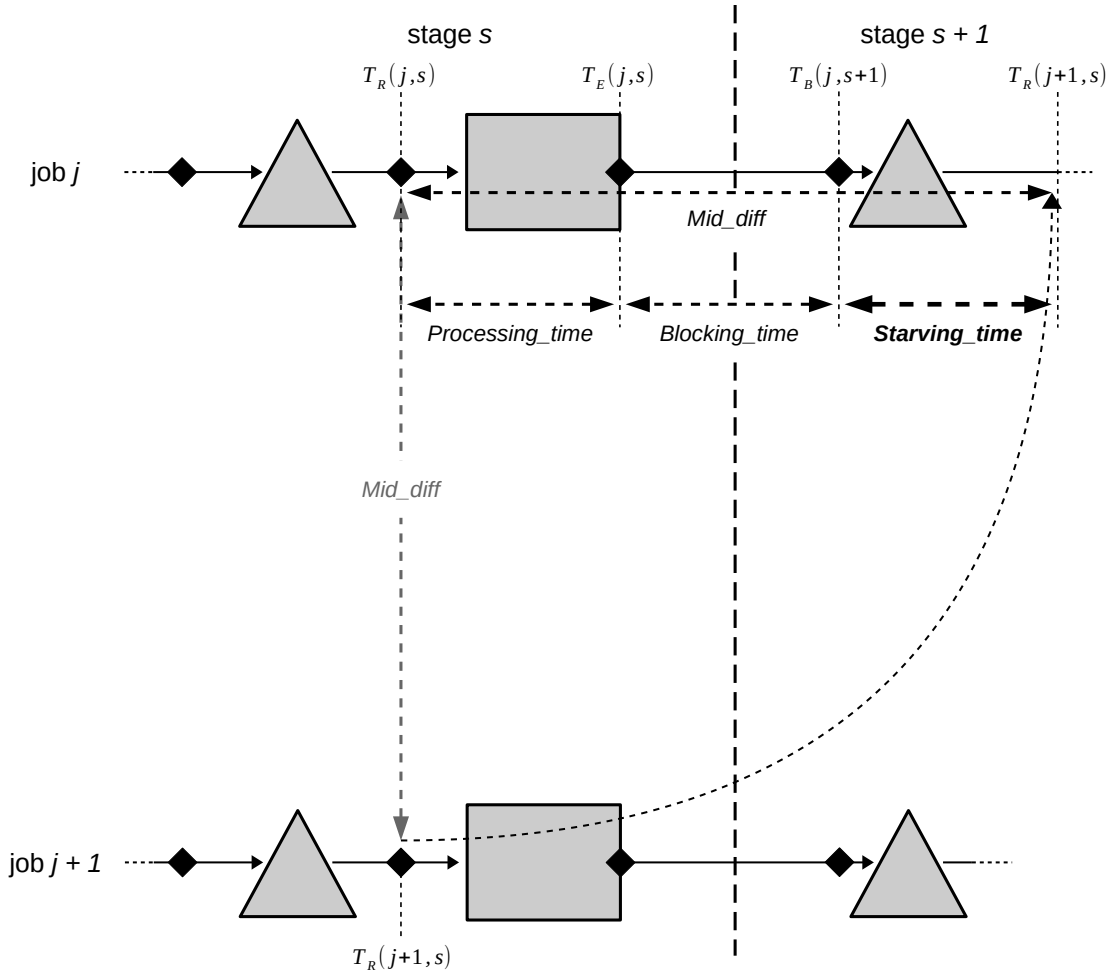


Fig. 4.7 Starving time computation scheme

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 (see figure 4.8). So, an alternative formula to compute Starving_time of job j in stage s is

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

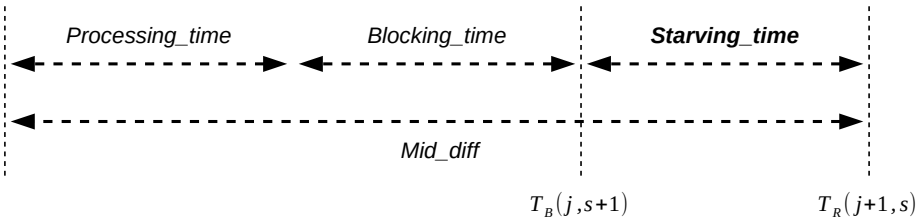


Fig. 4.8 Starving time alternative formula

Trend KPIs

Trend KPIs are calculated as ratios of CCI KPIs (see figure 4.9). 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 over time.

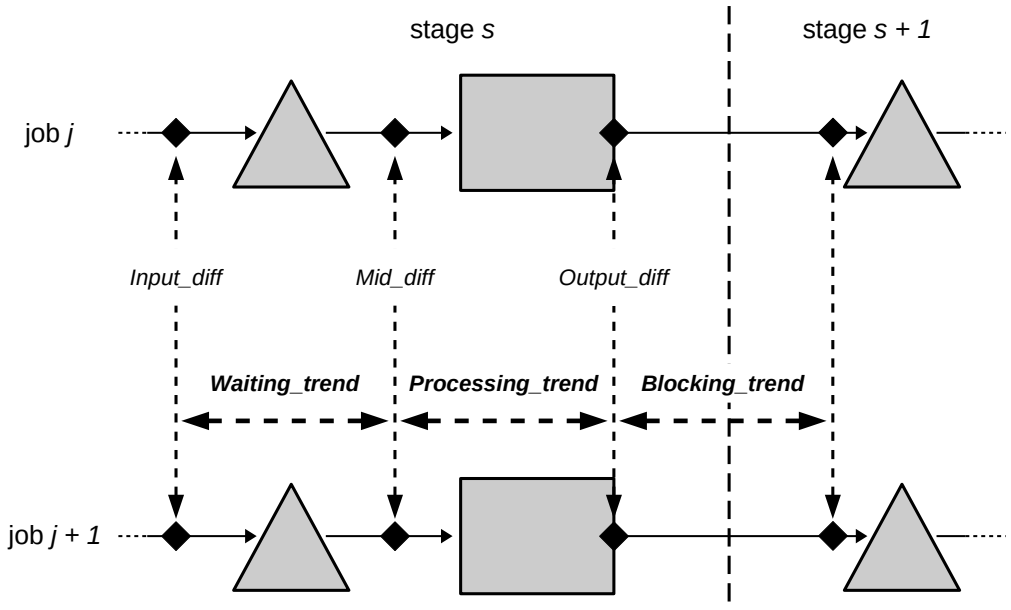


Fig. 4.9 Trend KPI computations scheme

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

Simulated systems

To observe the indicator behaviors, simulations are built to replicate a SSL having the characteristics described in section 3.2. Virtual sensors have been placed accordingly to the three sensor configuration shown in section 3.3. The sensor output is an event log having the field structure defined in chapter 4.
Concerning the software tools:

- Models are written and run with *simmer* [20], a process-oriented discrete-event simulator.
- 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 [21].
- Plots are printed using *ggplot2*, which is also part of the *tidyverse* collection.

simmer and *tidyverse* are open-source packages for R language [22].

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 by 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 to 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 $\Downarrow$	$\{1\}$	$\{1,3\}$	$\{1,3,6\}$

5.2 Rolling windows parameters

5.3 Plot structure

The KPIs described in section 4.3 are 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

This chapter aims to present a general, yet as much as possible complete collection of KPI behaviors, showing the most meaningful simulation results. For the sake of brevity, Basic KPIs plots are not discussed: the analysis would be too lengthy and dispersive, and they would be difficult to visualize since the data noise makes them barely understandable. Therefore, only RW KPIs are considered.
Unless otherwise specified, the RW KPIs are computed on windows with parameters $f = 100$ and $l = 100$.

6.1 Processing time variation

A processing time variation not only causes a change in Processing_time basic and derived KPIs in correspondence with the changing stage, but also influences different KPIs in all other stages, upstream and downstream the changing stage.
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

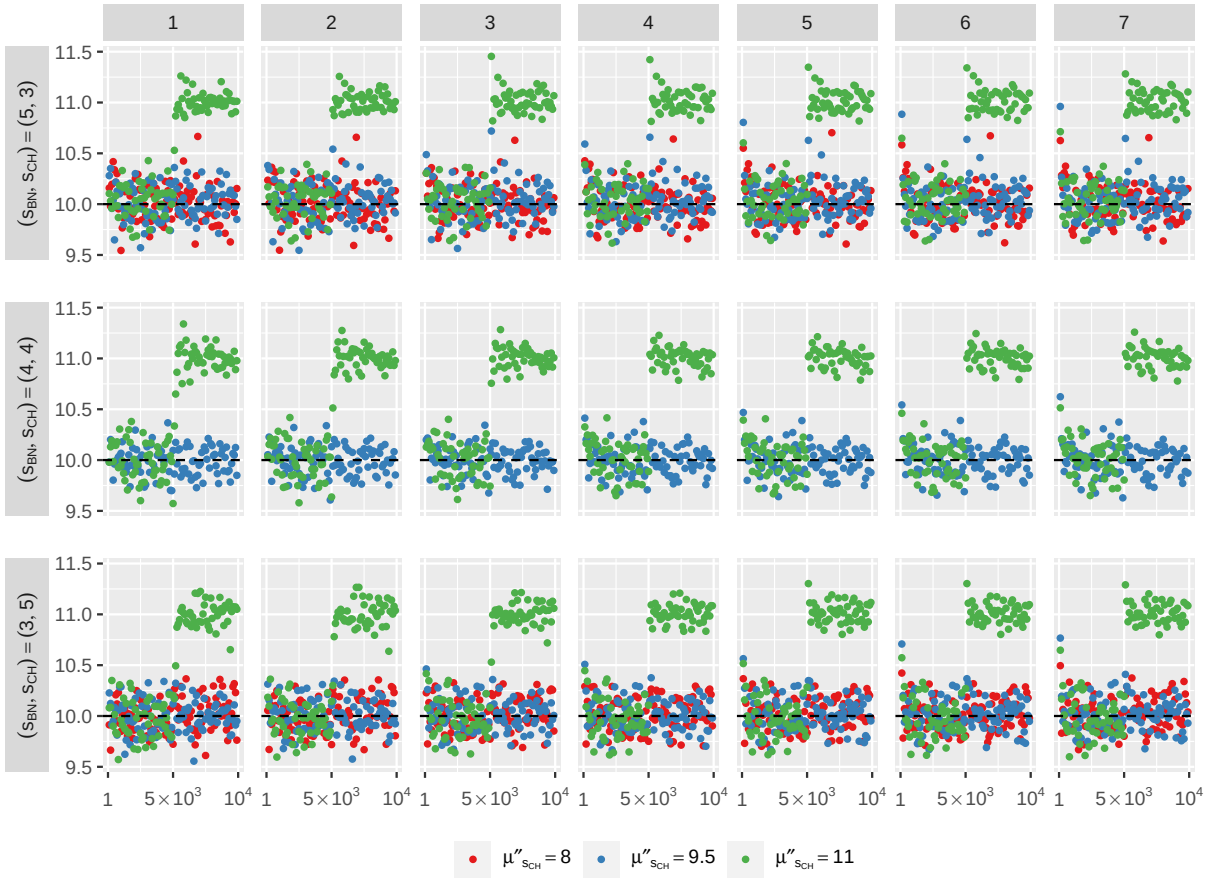


Fig. 6.1 Mid diff RW KPI considering three mean processing time increase levels $\mu''_{s_{CH}}$ and three bottleneck - changing stage configurations (s_{BN}, s_{CH}). Models with $cl_s = 6$.

Figure 6.1 shows Mid_diff_MEAN RW KPI behavior when a processing time growth occurs[1]. It can be deduced that

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

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

- 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.

- The value taken by average Mid_diff does not to depend on the stage position with respect to the changing stage s_{CH} or the bottleneck s_{BN}: it behaves similarly in all stages.
  This is visible comparing models having different bottleneck-changing stage configurations (s_{BN}, s_{CH}).

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). 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

[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 can be omitted.

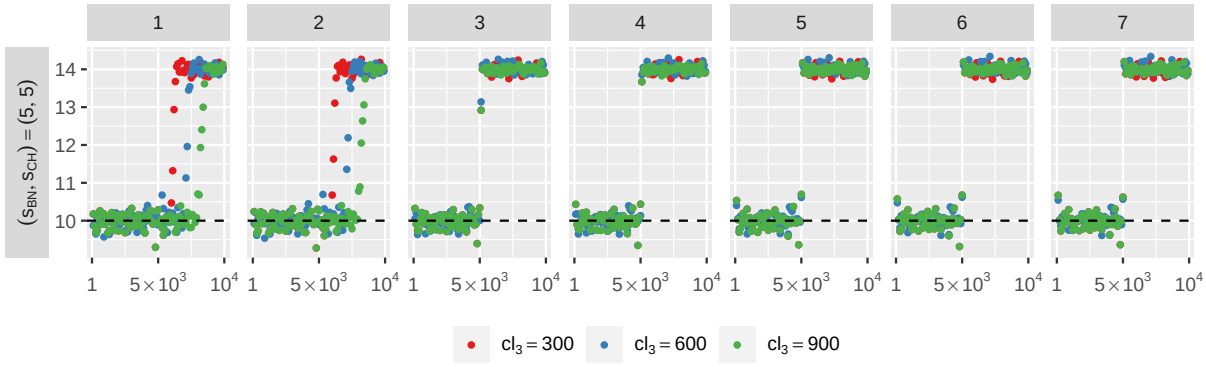


Fig. 6.2 Mid diff RW KPI variation delays in case of a mean processing time increase $\mu''_{s_{CH}}$ considering different buffer capacity limits cl_3. Models with configuration $(s_{BN}, s_{CH}) = (5,3)$ and processing time increase $\mu''_{s_{CH}} = 14$.

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

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.

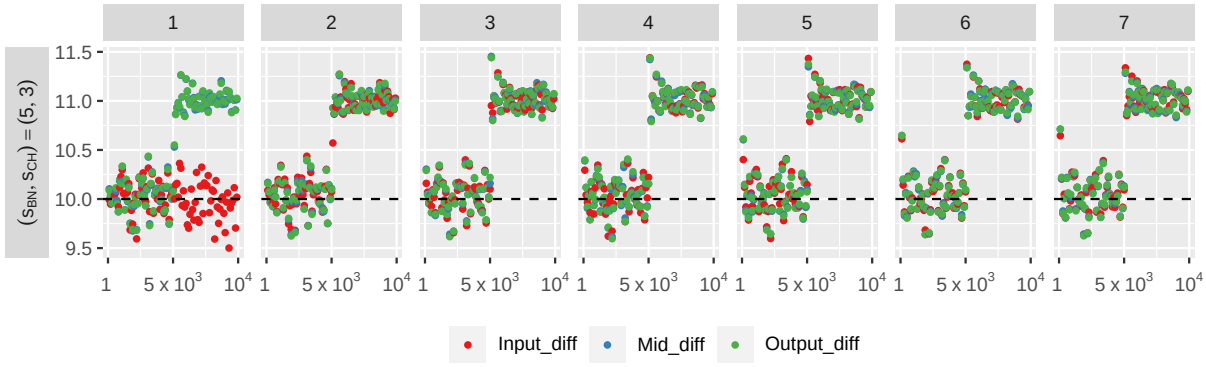


Fig. 6.3 Different CCI RW KPIs behaviors compared . Models with configuration $(s_{BN}, s_{CH}) = (4,4)$, $cl_s = 6$, and $\mu''_{s_{CH}} = 11$.

Input_diff 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.

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

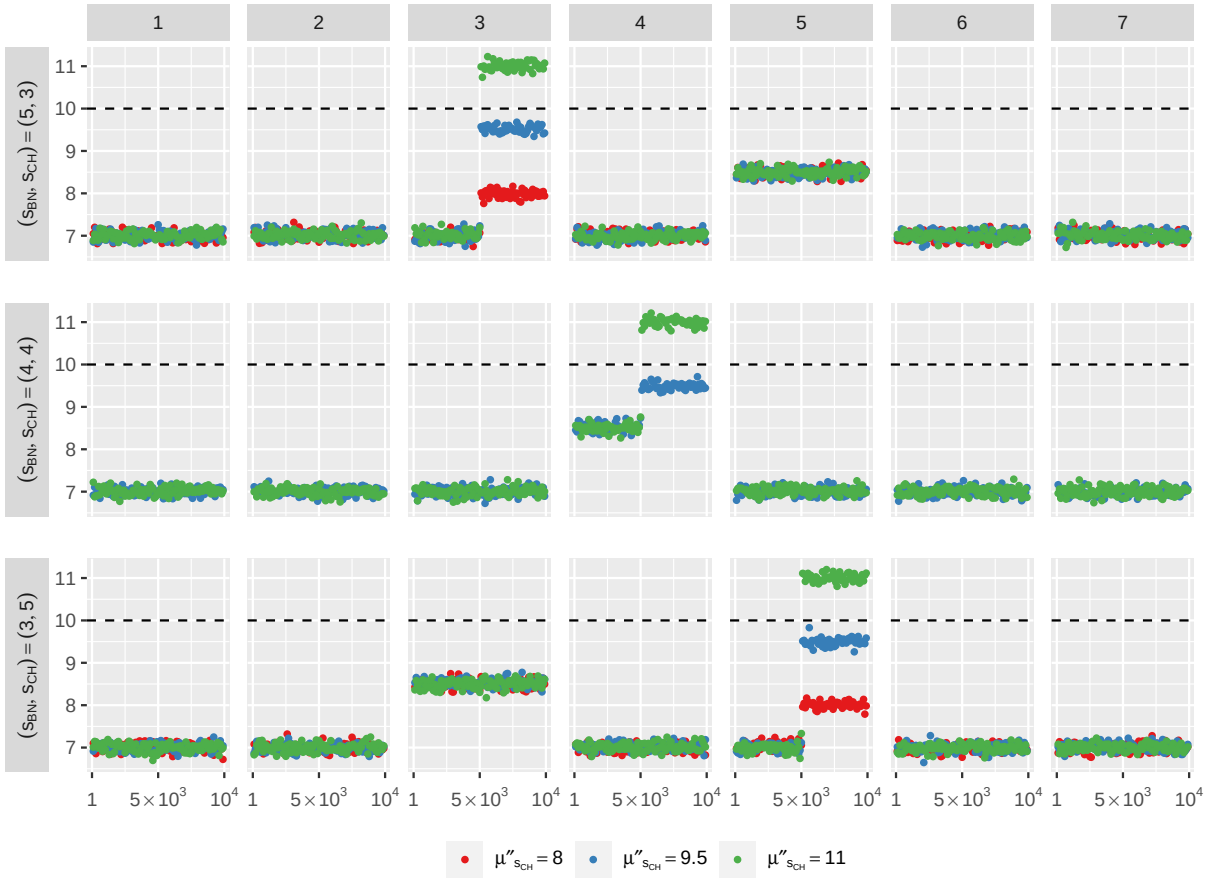


Fig. 6.4 Processing time RW KPI behavior considering three mean processing time increase levels $\mu''_{s_{CH}}$ and three bottleneck - changing stage configurations (s_{BN}, s_{CH}). Models with $cl_s = 6$.

Figure 6.4 shows Processing_time_MEAN RW KPI behavior when the production capacity of a stage reduces.

- If a resource loses production capacity, average Processing_time of the changing stage grows, while average Processing_time of all other stages does not change.
  This is visible comparing the behavior of Processing_time in correspondence with stage the changing s_{CH} (where it increases) with its behavior in other stages (where it does not change).

- If a resource loses production capacity, the variation of changing stage average Processing_time is such that the KPI value settles around the new average processing time of the stage.
  This is visible in each changing stage s_{CH} plot.

- Average Processing_time variation does not depend on the stage position with respect to the changing stage s_{CH} or the bottleneck s_{BN}.
  This is visible comparing average Processing_time behavior in the three different bottleneck - changing stage configurations (s_{BN}, s_{CH}).

- Average Processing_time variation does not depend on the system stability.
  This is visible comparing average Processing_time behavior in models with $\mu''_{s_{CH}} = 8$ and $\mu''_{s_{CH}} = 9.5$ (where the system remains stable) with its behavior in model with $\mu''_{s_{CH}} = 11$ (where the system becomes unstable).

- Average Processing_time of stage s always stabilizes around the average time needed by stage s resource to process a job.
  This is visible in all models, before and after the variation occurring when Case ID reaches value 5000.

To summarize, the graphs show that the average Processing_time of a stage is constantly aligned with the average processing time of that stage, even when the stage production capacity decreases.

Processing time decrease effects on Processing_time KPI

Figure 6.5 shows Processing_time_MEAN RW KPI behavior when the production capacity of a stage increases. Looking at these plots it can be concluded that it is possible to extend the observations made on average Processing_time behavior in case of production capacity decrease to production capacity variations in general. Indeed, the only difference with the production capacity decrease case is that average Processing_time of the changing stage s_{CH} reduces when production capacity increases to keep aligned with the average processing time of that stage.

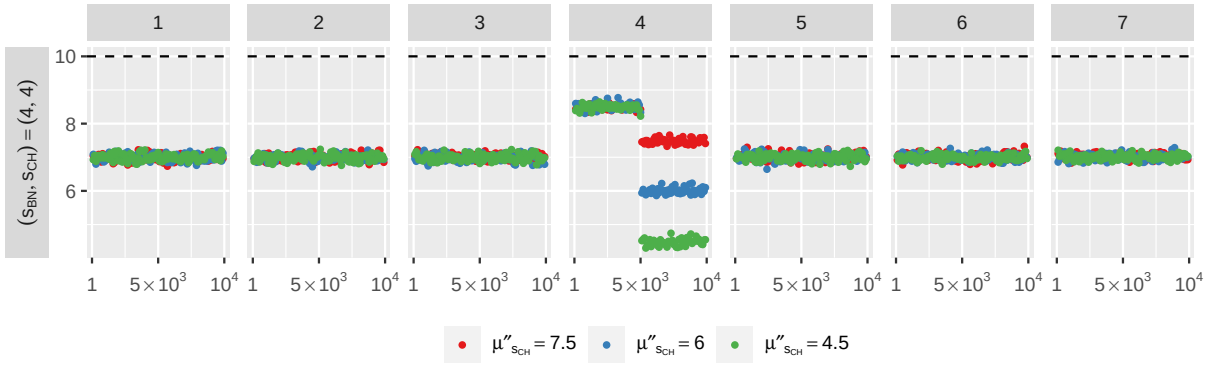


Fig. 6.5 Processing time RW KPI behavior considering three different mean processing time decrease levels $\mu_{s_{CH}}$. Models with configuration $(s_{BN}, s_{CH}) = (4,4)$, and $cl_s = 6$.

Utilization KPI

Processing_time alone is useful to monitor stage production capacities, but does not give insights about how much stages work with respect to the line throughput. To obtain this information, Processing_time_MEAN with Mid_diff_MEAN are combined to calculate Utilization KPI. Figure 6.6 shows Utilization KPI behavior when the production capacity of a stage reduces. This KPI behavior is similar to average Processing_time behavior, with some important differences

- If a resource loses production capacity, but the system remains stable, Utilization of the changing stage grows, while Utilization of all other stages does not change.
  This is visible in models with $\mu''_{s_{CH}} = 8$ and $\mu''_{s_{CH}} = 9.5$, comparing the behavior of Utilization in correspondence with stage the changing s_{CH} (where it increases) with its behavior in other stages (where it does not change).

- If a resource loses production capacity to the point that the system becomes unstable, Utilization of the changing stage grows and settles to value 1, while Utilization of all other stages reduces.
  This is visible in models with $\mu''_{s_{CH}} = 11$, comparing the behavior of Utilization in correspondence with stage the changing s_{CH} (where it increases) with its behavior in other stages (where it decreases).

- Utilization variation does not depend on the stage position with respect to the changing stage s_{CH} or the bottleneck s_{BN}. This is visible comparing Utilization behavior in the three different bottleneck - changing stage configurations (s_{BN}, s_{CH}).

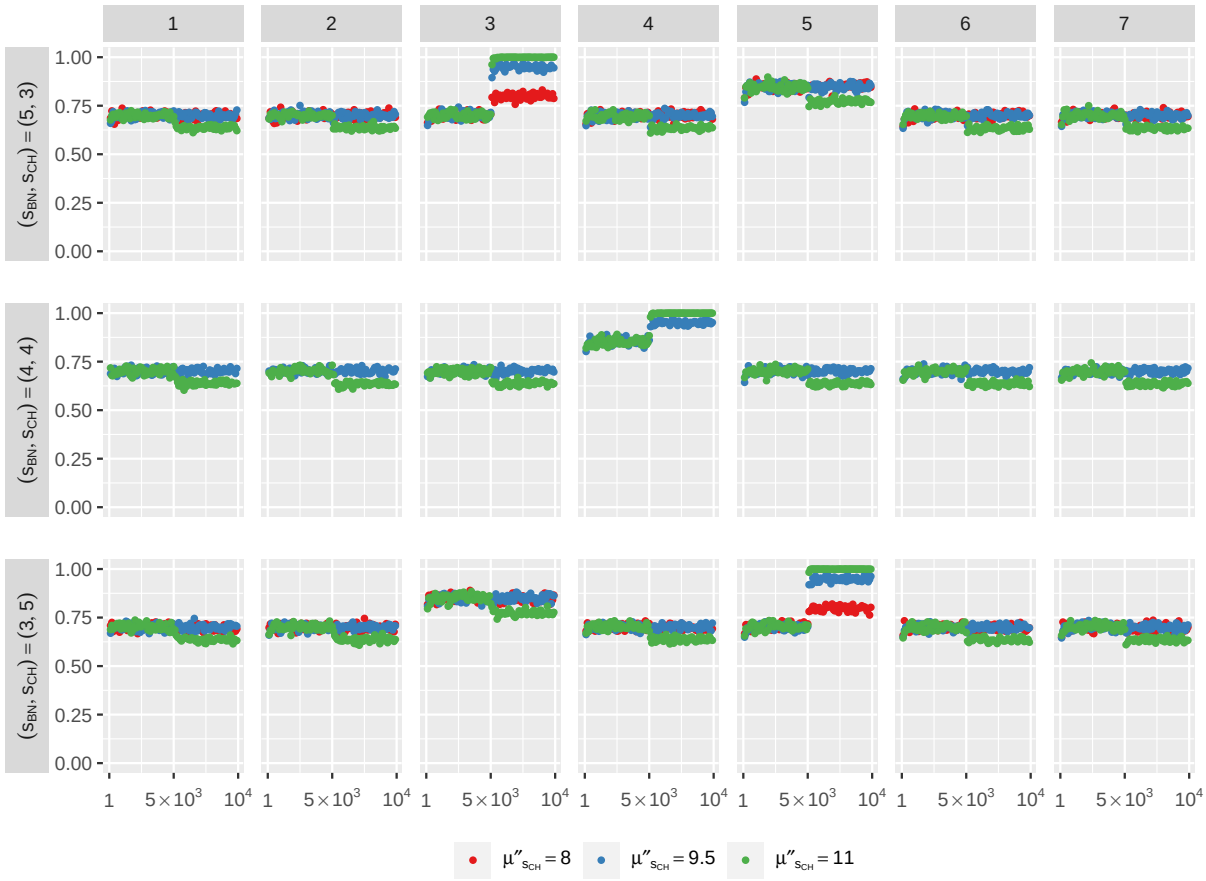


Fig. 6.6 Utilization KPI behavior considering three mean processing time increase levels $\mu''_{s_{CH}}$ and three bottleneck - changing stage configurations (s_{BN}, s_{CH}). Models with $cl_s = 6$.

- Values taken by Utilization are limited to the range $[0, 1]$.
  This is visible in all models, before and after the variation occurring when Case ID reaches value 5000.
- Utilization of stage s stabilizes around the average utilization of stage s resource.
  This is visible in all models, before and after the variation occurring when Case ID reaches value 5000.

Utilization KPI follows the theoretical behavior of resource utilization. Indeed, the utilization of a resource is equal to the ratio between throughput and production capacity of the resource, or in other words, it is equal to the ratio between the resource processing time and the system cycle time. Therefore, if the system becomes unstable, cycle time grows aligning with the bottleneck processing time, and so the resource utilization in the bottleneck reaches 100%. Instead, after all

other stages have synchronized with the bottleneck lowering their throughput, their utilization reduces.
Even if the relative plots are not shown, it is possible to extend the observations made on Utilization behavior in case of production capacity decrease to production capacity variations in general, as with average Processing_time. The only difference with the production capacity decrease case is that Utilization of the changing stage s_{CH} reduces when production capacity increases to keep always aligned with utilization of that stage.

6.1.3 Blocking_time, Starving_time and respective Stage State KPIs

Blocking_time and Starving_time KPIs are analyzed together because of their strong relationship. As a matter of fact, their behavior is specular, since they both depend on the resource utilizations and on the buffer capacities. This subsection starts showing and discussing these KPI general behaviors and the effects of a processing time variation on them. Then, Bloking_prob and Starving_prob, which are respectively Blocking_time and Starving_time derived Stage State KPIs, are commented.

Processing time increase and reduction effects on Blocking_time and Starving_time KPIs

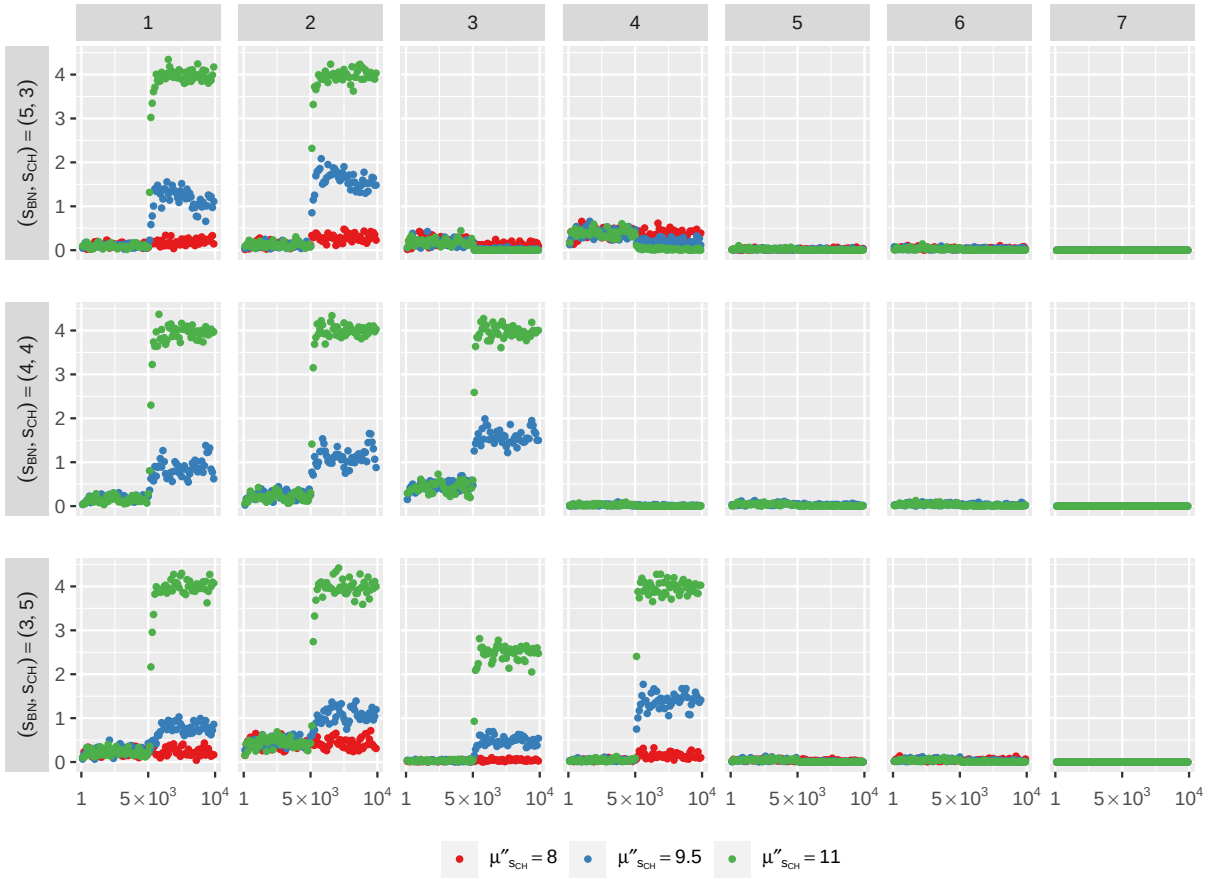


Fig. 6.7 Blocking time RW KPI considering three mean processing time increase levels $\mu''_{s_{CH}}$ and three bottleneck - changing stage configurations (s_{BN}, s_{CH}). Models with $cl_s = 6$.

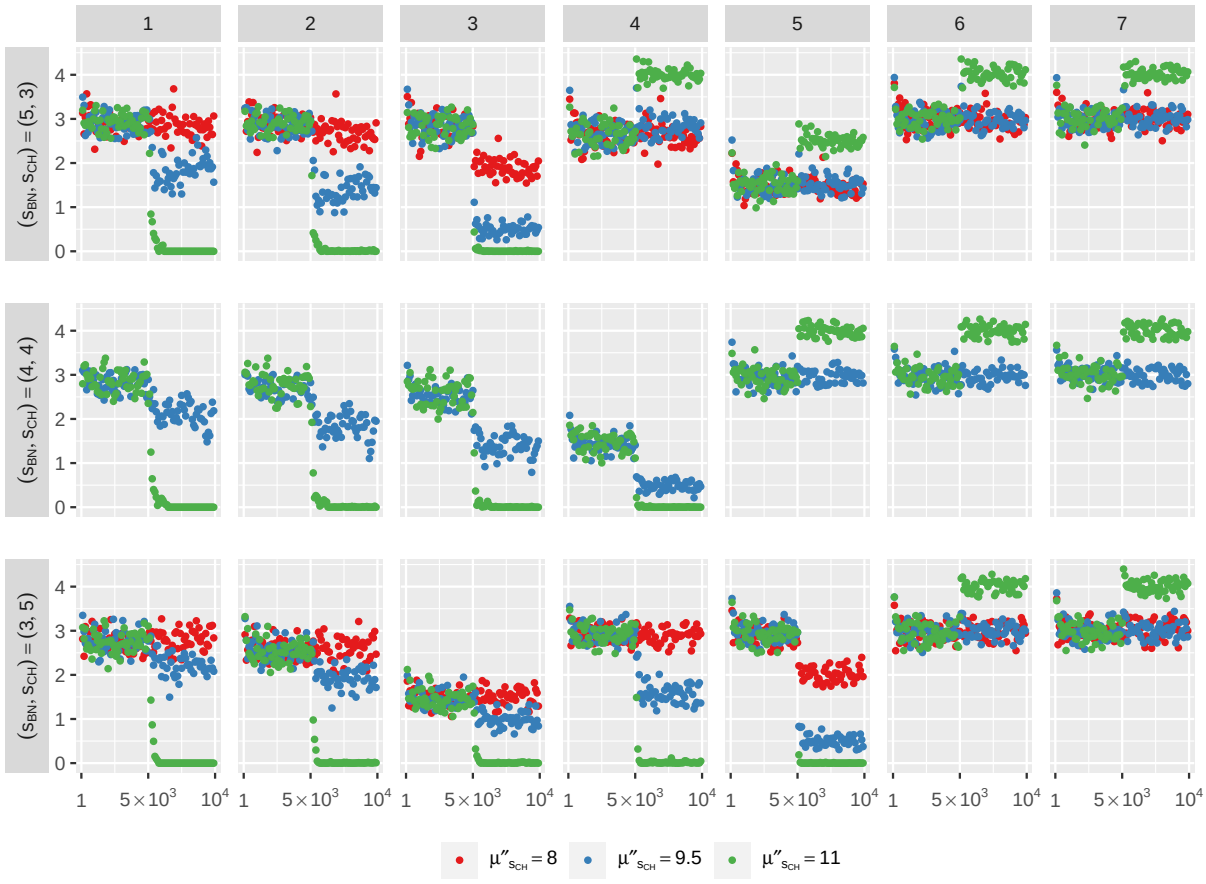


Fig. 6.8 Starving time RW KPI considering three processing time increase levels $\mu''_{s_{CH}}$ and three bottleneck - changing stage configurations (s_{BN}, s_{CH}). Models with $cl_s = 6$.

Figures 6.7 and 6.8 show, respectively, Blocking_time_MEAN and Starving_time_MEAN RW KPI behaviors when the production capacity of a stage reduces. Looking at these plots it can be said that

- If a resource loses production capacity to the point that the system becomes unstable:
  - In stages upstream the changing stage s_{CH}, average Blocking_time increases and settles around value I_s, while average Starving_time decreases and settles around value 0.
  - In correspondence of the changing stage s_{CH}, both average Blocking_time and average Starving_time decrease and settle around value 0.

  - In stages downstream the changing stage s_{CH}, average Blocking_time decreases and settles around value 0, while average Starving_time increases and settles around value I_s.

  This is visible in models with any configuration (s_{BN}, s_{CH}) having after-change mean processing time value $\mu''_{s_{CH}} = 11$.
  I_s is equal to the difference between average Mid_diff and average Processing_time of stage s. In other words, it is a limit that depends on the cycle time as seen in stage s, and on stage s production capacity.

- If a resource loses production capacity but the system remains stable:

  - In stages upstream the changing stage s_{CH}, average Blocking_time increases towards I_s, while average Starving_time decreases towards 0.
  - In correspondence of the changing stage s_{CH}, both average Blocking_time and average Starving_time decrease towards 0.
  - In stages downstream the changing stage s_{CH}, average Blocking_time decreases towards 0, while average Starving_time increases towards I_s.

  This is visible in models with any configuration (s_{BN}, s_{CH}) having after-change mean processing time value $\mu''_{s_{CH}} = 8$ or $\mu''_{s_{CH}} = 9.5$.

- Average Blocking_time and average Starving_time variations in a certain stage s are less and less relevant the further stage s is from the changing stage s_{CH}, both upstream and downstream s_{CH}.

- Average Blocking_time and average Starving_Queue variations are wider the more significant is the increase of processing time.

- Average Blocking_time of stage s stabilizes around the average time a job has to wait in stage s resource after having been processed.
  Average Starving_time of stage s stabilizes around the average time passing between the exit of a job from stage s resource and the entrance of a new one.
  This is visible in all models, before and after the variation occurring when Case ID reaches value 5000.

Blocking_time KPI and Starving_time KPI follow the theoretical behavior of, respectively, blocking and starvation duration. Indeed, in a stable system, blocking is high in stages upstream

the bottleneck, and much lower in stages downstream and in correspondence. Conversely, starvation is lower in stages upstream and in correspondence with the bottleneck, and higher downstream. When the system becomes unstable, blocking and starvation behaviors are led to the extreme

- Upstream the bottleneck, when buffers become saturated, blocking causes a delay in resource releases, that last until the bottleneck has finished to process a job; so, blocking duration of a certain resource is equal to the time difference between the processing finish of a job in the bottleneck and the processing finish of a job in the considered stage.
  Downstream the bottleneck, since buffers tend to be completely empty, blocking duration drops to 0.
- Downstream and in correspondence of the bottleneck, starvation causes resources idleness 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.
  Upstream the bottleneck, since buffers tend to be completely full, starvation duration drops to 0.

All the observations that have been done regarding these KPIs considering production capacity reductions are similarly valid also in case of production capacity growth. Figure 6.9 displays average Blocking_time and average Starving_time in case a resource processing time decreases. The only difference with the processing time increase case is that average Blocking_time and average Starving_time behaviors are inverted when the variation occurs

- In stages upstream the changing stage s_{CH}, average Blocking_time decreases, while average Starving_time increases.
- In correspondence of the changing stage s_{CH}, both average Blocking_time and average Starving_time increase.
- In stages downstream the changing stage s_{CH}, average Blocking_time increases, while average Starving_time decreases.

Blocking_time and Starving_time KPIs considering different buffer levels

Previously only one buffer capacity limit ($cl_s = 6$) was taken into account in the study. However, Blocking_time and Starving_time KPIs are also influenced by the buffer capacity, and it is

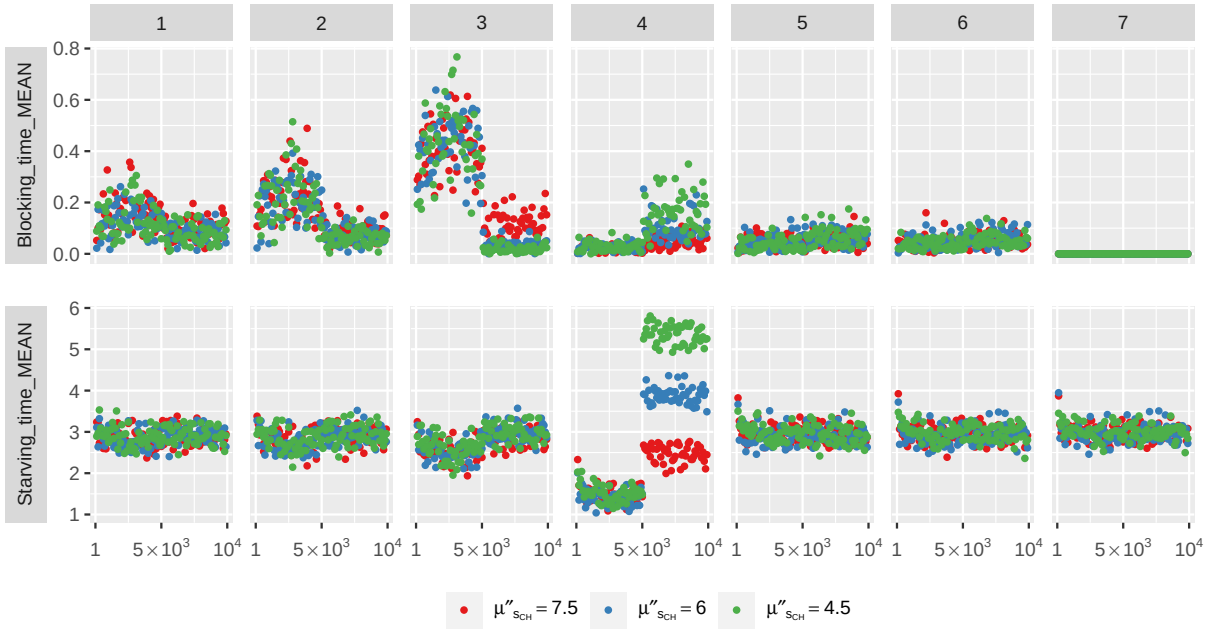


Fig. 6.9 Blocking time and Starving time KPI behaviors considering three different mean processing time decrease levels $\mu''_{s_{CH}}$. Models with configuration $(s_{BN}, s_{CH}) = (4,4)$ and $cl_s = 6$.

interesting to analyze their behavior considering different capacity levels. Figure 6.10 shows average Blocking_time and average Starving_time behaviors in case of a processing time increase, but considering models having three different buffer capacity limits cl_s.
When a stage loses production capacity

- The variation extents of average Blocking_time and average Starving_time are wider the lower the buffer capacity limit is
- The variation extents of average Blocking_time and average Starving_time are narrower the higher the buffer capacity limit is

It is particularly visible comparing the model with $cl_s = 3$ and the one with $cl_s = 6$.
This happens because smaller buffers are on average easily saturated, causing longer resource blockings in previous stages and reducing starvation. When buffers are bigger the opposite verifies.

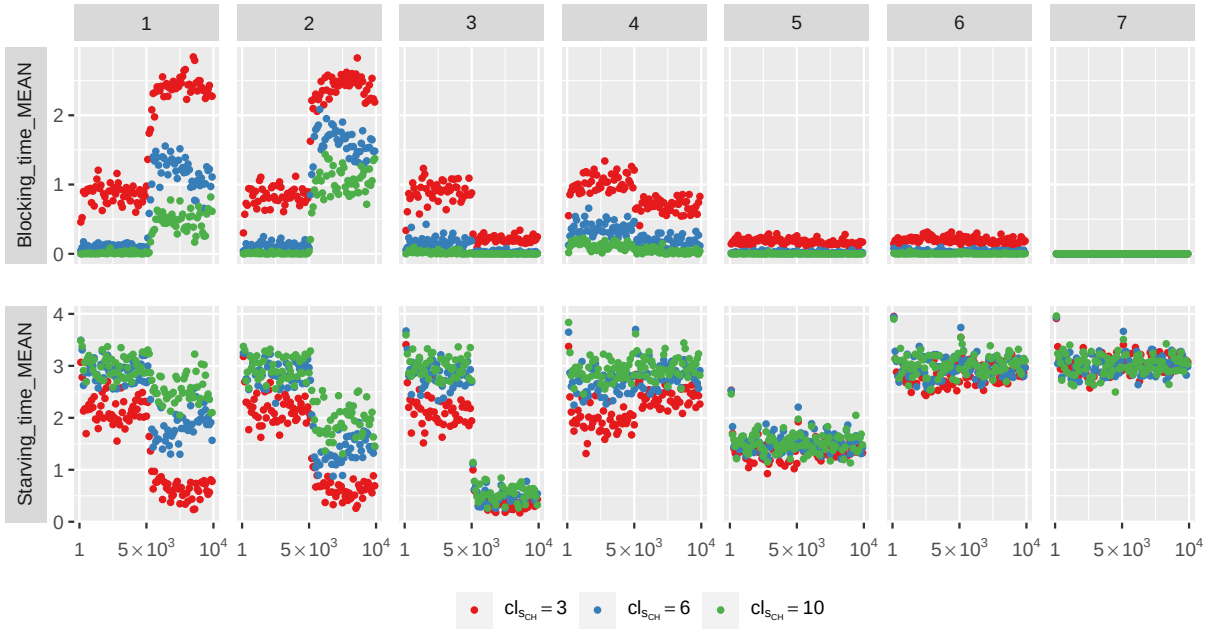


Fig. 6.10 Blocking time and Starving time RW KPI behaviors in case of mean processing time increase $\mu''_{s_{CH}}$ considering different buffer capacity limits cl_s. Models with configuration $(s_{BN}, s_{CH}) = (5,3)$ and $\mu''_{s_{CH}} = 9.5$.

Value I_s behavior and Blocking_time and Starving_time relative proportions

It is noteworthy to explain that value I_s [3] is not simply a superior limit for average Blocking_time or average Starving_time, but actually it is always equal to their sum. As a matter of fact, the sum of Blocking_time and Starving_time of stage s is always equal to the difference between Mid_diff and Processing_time of stage s because of how these KPIs are defined (see subsection 4.3.2).

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

Theoretically, this is reasonable given the characteristics of the considered systems: since machine breakdowns are assumed to never occur, resources can find themselves in three possible states, which are seized and working state (processing), seized and not working state (blocking), and not seized state (starvation). Sum of the total time spent by a resource in these states gives the whole available time of the resource, as, analogously, Processing_time, Blocking_time, and Starving_time add up matching Mid_diff. Figure 6.11 is helpful to understand how I_s behaves.

[3] Remember that I_s was defined as the difference between average Mid_diff and average Processing_time of stage s

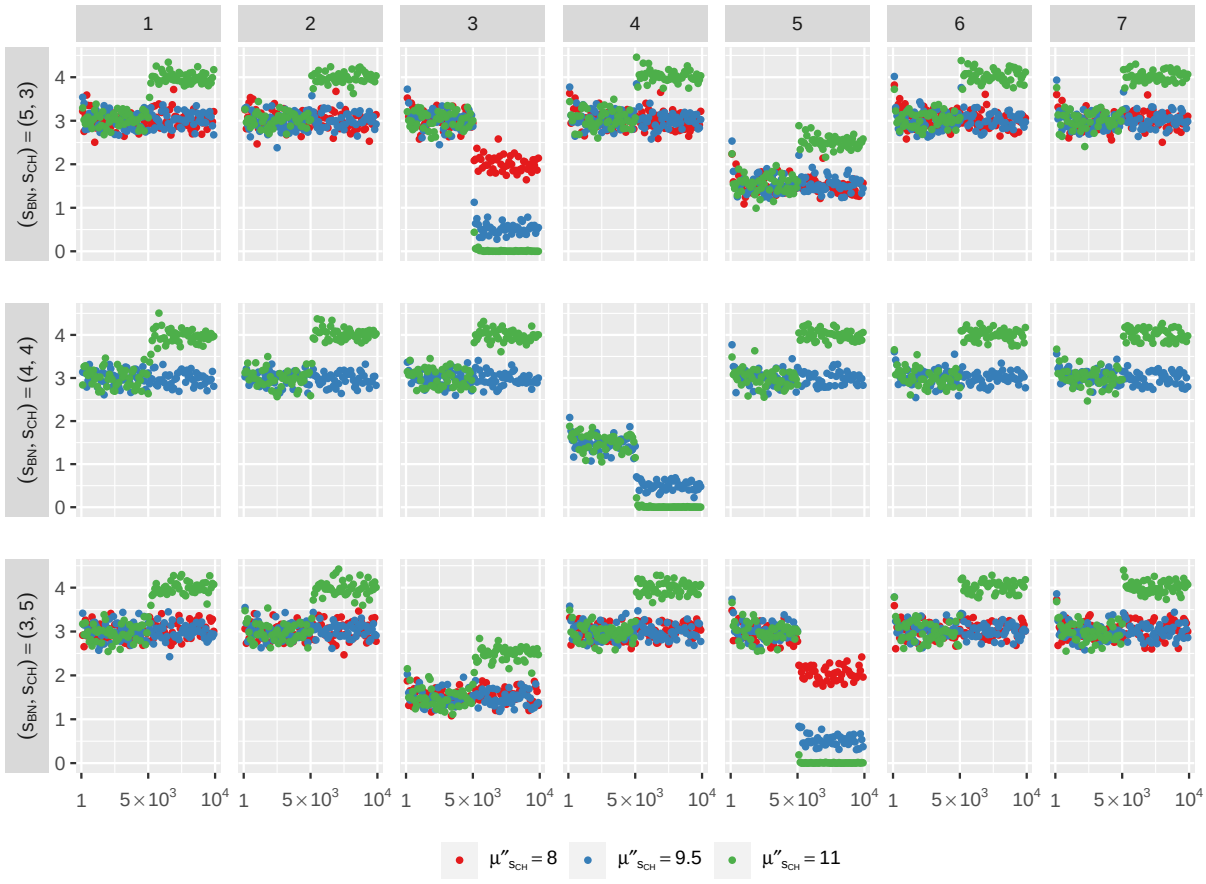


Fig. 6.11 Value I_s considering three different mean processing time increase levels $\mu''_{s_{CH}}$ and three bottleneck - changing stage configurations (s_{BN}, s_{CH}). Models with $cl_s = 6$.

- If a production capacity variation occurs, but the system remains stable, average I_s changes only in correspondence of the changing stage s_{CH} (it reduces if $\mu''_{s_{CH}} > \mu'_{s_{CH}}$, increases if $\mu''_{s_{CH}} < \mu'_{s_{CH}}$).
  It is visible in models with $\mu''_{s_{CH}} = 8$ or $\mu''_{s_{CH}} = 9.5$.
- If a production capacity variation occurs and the system becomes unstable, average I_s drops to 0 in correspondence with the changing stage s_{CH}, and grows in all other stages.
  It is visible in models with $\mu''_{s_{CH}} = 11$.

Since I_s is also the sum of average Blocking_time and average Starving_time, this means that the sum of these KPIs, if the system is stable, is constant in all stages except for the changing stage s_{CH}. Instead, if the system becomes unstable, the sum reduces to 0 in s_{CH} and grows in all

other stages. At this point, it is interesting to visualize the relative proportions of Blocking_time and Starving_time in this sum. Figure 6.12 shows that

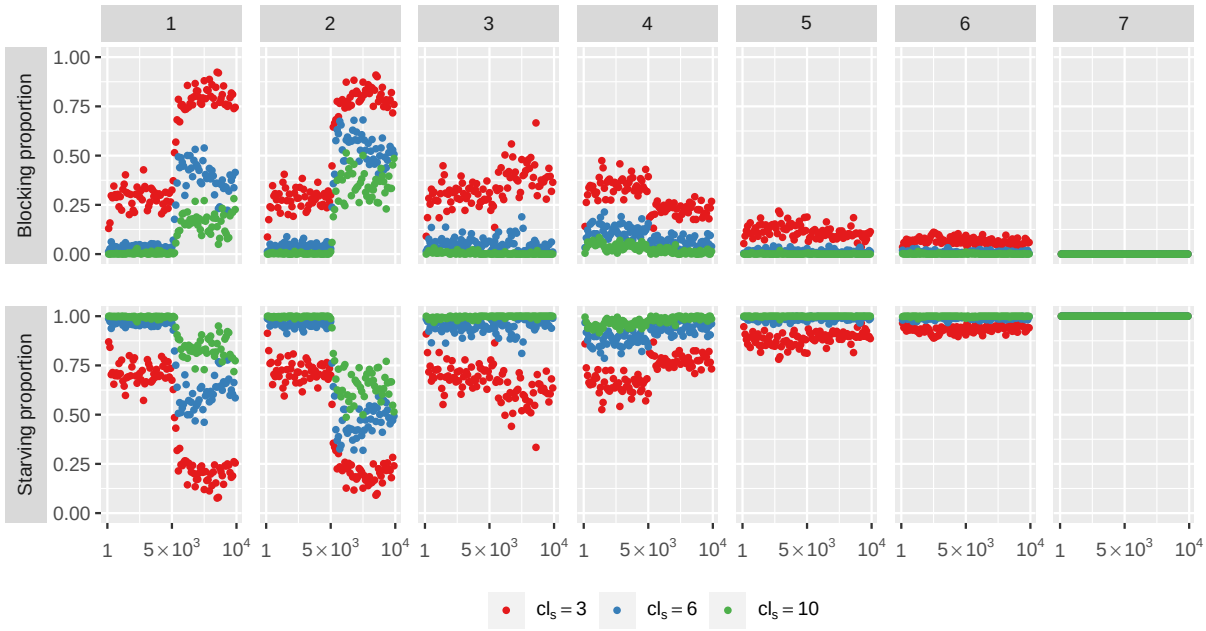


Fig. 6.12 Average Blocking time and Starving time relative proportions considering different buffer capacity limits cl_s. Models with configurations $(s_{BN}, s_{CH}) = (5,3)$, and after-change mean processing time $\mu''_{s_{CH}} = 9.5$.

- The contribution of average Blocking_time in I_s increases the smaller is the buffer.
- The contribution of average Starving_time in I_s reduces the smaller is the buffer.

Blocking_prob and Starving_prob KPIs

Like Processing_time, also Blocking_time and Starving_time can be represented in combination with Mid_diff. The obtained KPIs are Blocking_prob and Starving_prob. Figure 6.13 portrays an example of these KPIs. These KPIs indicate how much of the available time was spent by the resource without being utilized. The observations that can be made regarding these KPIs behavior are analogous to one done for Blocking_time and Starving_time.

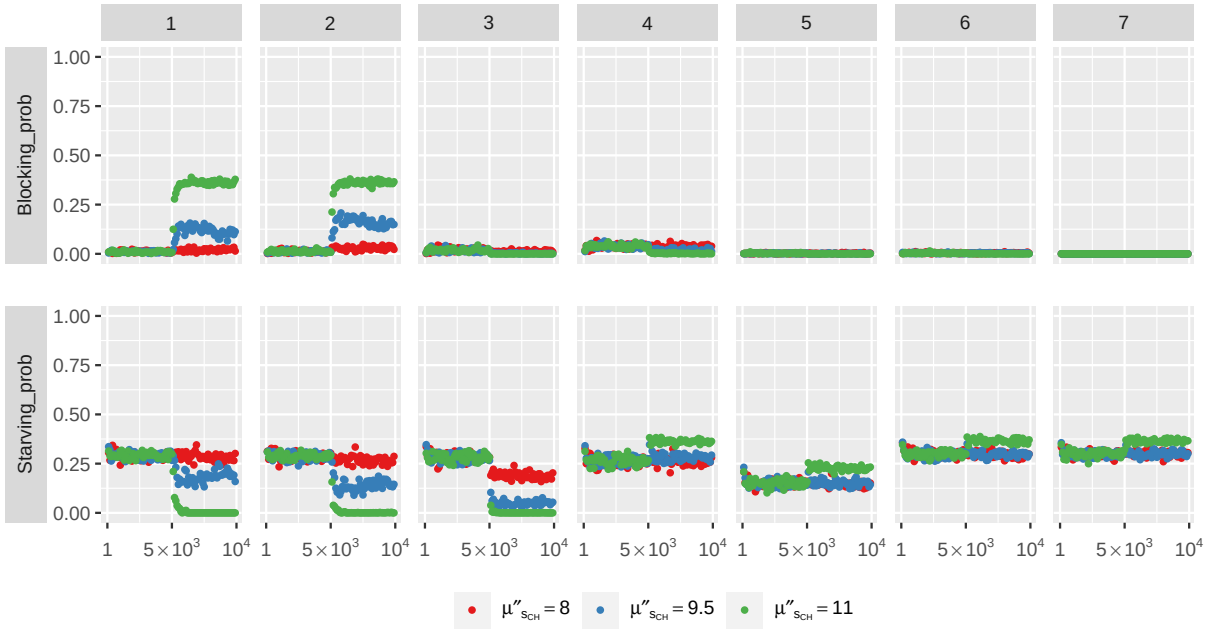


Fig. 6.13 Blocking prob and Starving prob KPI behaviors considering three processing time increase levels $\mu''_{s_{CH}}$. Models with configurations $(s_{BN}, s_{CH}) = (5,3)$ and $cl_s = 6$.

6.1.4 Waiting_time and Average_Queue KPIs

Blocking_time and Starving_time help to monitor the effects of a variation on the exploitation of resources in stages different from the changing stage; indirectly, these KPIs allow to get insights about the buffer saturation levels. However, to get a closer and more precise view of job queues statuses, it is necessary to resort to Waiting_time KPI. In this subsection, firstly Waiting_time and Average_Queue, which is its derived Stage State KPI, behaviors are discussed; then, at the end, the effects of having an unlimited buffer in the first stage are briefly addressed.

Processing time increase effects on Waiting_time KPI

Figure 6.14 the behavior of Waiting_time_MEAN RW KPI in case of a production capacity decrease in a stage.

- If a resource loses production capacity to the point that the system becomes unstable:
  - In stages upstream the changing stage s_{CH}, average Waiting_time increases and settles around value LW_s.

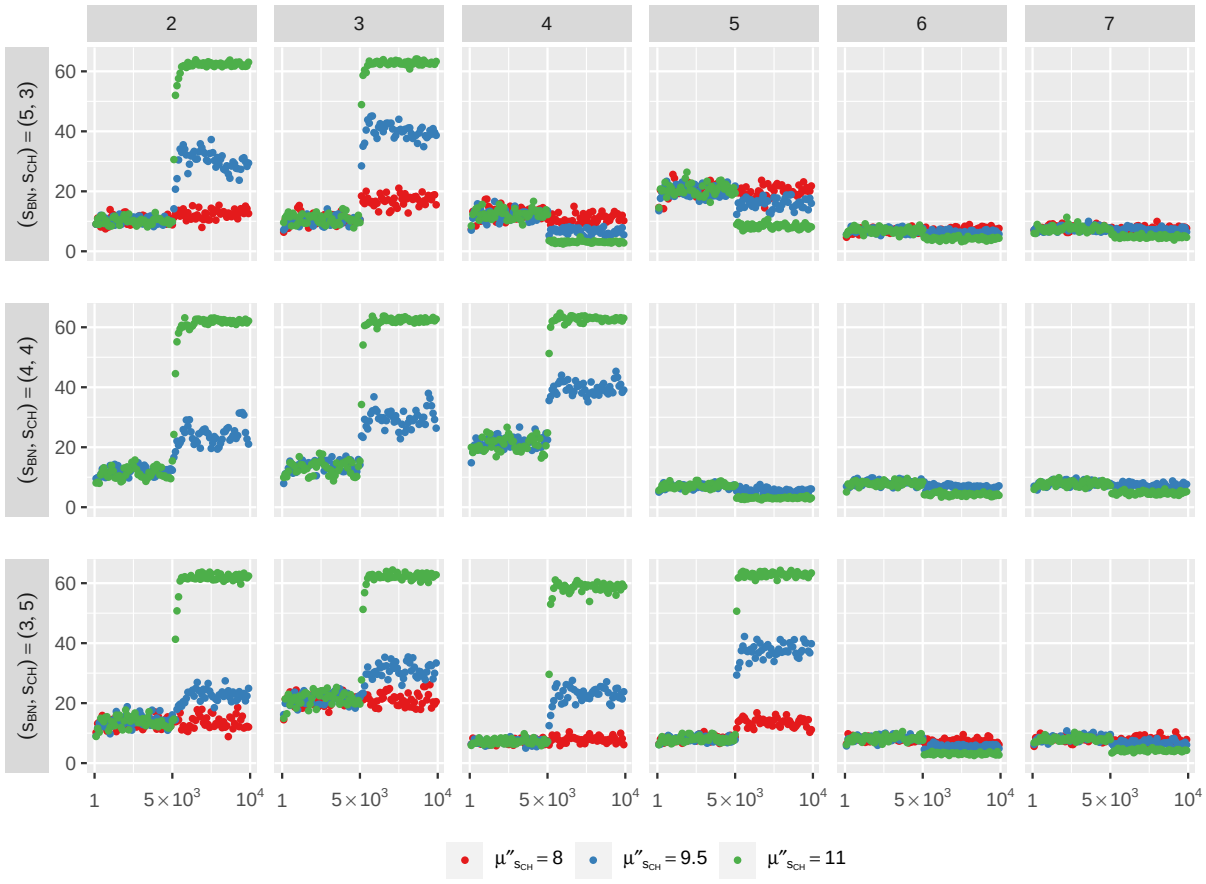


Fig. 6.14 Waiting time RW KPI behavior considering three processing time increase levels $\mu''_{s_{CH}}$ and three bottleneck - changing stage configurations (s_{BN}, s_{CH}). Models with $cl_s = 6$.

- In correspondence of the changing stage s_{CH}, average Waiting_time increases and settles around value LW_s.
- In stages downstream the changing stage s_{CH}, average Waiting_time decreases towards 0.

This is visible in models with any configuration (s_{BN}, s_{CH}) having after-change mean processing time value $\mu''_{s_{CH}} = 11$.
LW_s is equal to the product of average Input_diff and the maximum buffer capacity of stage s, cl_s. In other words, it is a limit that depends on the cycle time as seen in stage s, and on stage s buffer capacity.

- If a resource loses production capacity but the system remains stable:
  - In stages upstream the changing stage s_{CH}, average Waiting_time increases.

  - In correspondence of the changing stage s_{CH}, average Waiting_time increases.
  - In stages downstream the changing stage s_{CH}, average Waiting_time decreases.

  This is visible in models with any configuration (s_{BN}, s_{CH}) having after-change mean processing time value $\mu''_{s_{CH}} = 8$ or $\mu''_{s_{CH}} = 9.5$.

- Average Waiting_time variations are wider the more significant is the increase of processing time.
- Average Waiting_time variations in a certain stage s are less and less relevant the further stage s is from the changing stage s_{CH}, both upstream and downstream s_{CH}.
- Average Waiting_time of stage s stabilizes around the average time a job has to wait in stage s buffer before seizing the resource.
  This is visible in all models, before and after the variation occurring when Case ID reaches value 5000.

Average Waiting_time of stage s follows the theoretical behavior of the average time a job spends in stage s buffer. Indeed, the time a job has to wait is longer in stages upstream the bottleneck and lower in stages downstream. Moreover, if the system becomes unstable, buffers in stages upstream the bottleneck get saturated, and the job queue is longer the higher the buffer capacity limit is. Since jobs in buffers, before seizing the resource, has to wait that all other jobs which have entered before them are processed [4], the time they have to wait is equal to the number of jobs in the queue before them (which is equal to the buffer limit, since it is full) multiplied by the job inter-arrival time in the queue, that is the cycle time.

Value LW_s and Average_Queue KPI

LW_s has been defined as a limit value for stage s average Waiting_time if the system becomes unstable, however its equation can be generalized to be also valid when the system works in a stable condition. Indeed, theory says jobs waiting time in a queue is always equal to the average cycle time multiplied by the average number of jobs in the queue. This means that average Waiting_time of stage s is always equal to the product of stage s average Input_diff [5] and the average number of jobs in stage s buffer. LW_s is just the value taken by average Waiting_time in stages upstream the bottleneck when a particular situation occurs, that is when system becomes

[4] Remember that FIFO was assumed to be the serving policy, see section 3.2

[5] Input_diff is the indicator of the cycle time as it is seen at buffer entrances

unstable.

This observation explains why Average_queue (i.e. the ratio between average Waiting_time and Input_diff, see subsection 4.3.2) indicates the average number of jobs in a buffer. Figure

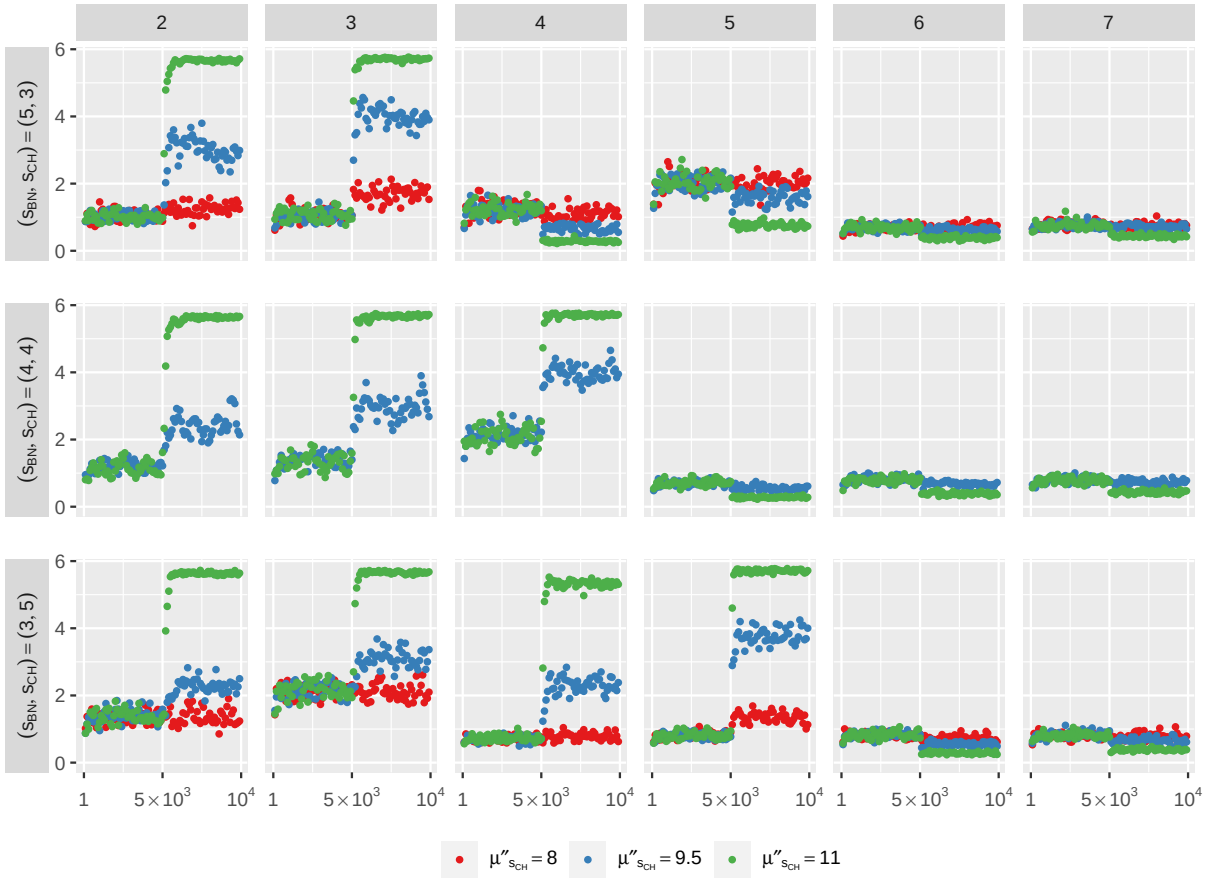


Fig. 6.15 Average Queue KPI behavior considering three processing time increase levels $\mu''_{s_{CH}}$ and three bottleneck - changing stage configurations (s_{BN}, s_{CH}). Models with $cl_s = 6$.

6.15 portrays Average_Queue considering the same models of figure 6.14. This KPI behavior is similar to the Waiting_time one

- If a resource loses production capacity to the point that the system becomes unstable:
  - In stages upstream the changing stage s_{CH}, Average_Queue increases and settles around the buffer capacity limit cl_s.
  - In correspondence of the changing stage s_{CH}, Average_Queue increases and settles around the buffer capacity limit cl_s.

  - In stages downstream the changing stage s_{CH}, Average_Queue decreases towards 0.

  This is visible in models with any configuration (s_{BN}, s_{CH}) having after-change mean processing time value $\mu''_{s_{CH}} = 11$.

- If a resource loses production capacity but the system remains stable:

  - In stages upstream the changing stage s_{CH}, Average_Queue increases towards the buffer capacity limit cl_s.
  - In correspondence of the changing stage s_{CH}, Average_Queue increases towards the buffer capacity limit cl_s.
  - In stages downstream the changing stage s_{CH}, Average_Queue decreases towards 0.

  This is visible in models with any configuration (s_{BN}, s_{CH}) having after-change mean processing time value $\mu''_{s_{CH}} = 8$ or $\mu''_{s_{CH}} = 9.5$.

- Average_Queue variations are wider the more significant is the decrease of processing time.

- Average_Queue variations in a certain stage s are less and less relevant the further stage s is from the changing stage s_{CH}, both upstream and downstream.

- Average_Queue of stage s stabilizes around the average number of jobs present in stage s buffer.
  This is visible in all models, before and after the variation occurring when Case ID reaches value 5000.

Waiting_time behavior considering different buffer sizes

As said before, Waiting_time and Average_Queue behaviors depend on buffer capacity limits. However, the buffer limitation does not affect these KPIs only when the system is unstable (i.e. when Waiting_time reaches its limit LW_s and Average_Queue settles around the buffer maximum capacity cl_s), but it can influence them also when the system is stable. For sake of brevity, since Waiting_time and Average_Queue behaviors are very similar, only plots of the first one are displayed. Figure 6.16 shows Waiting_time in case of production capacity loss, considering three different buffer limits cl_s. Since the influence of buffer capacity limits on Waiting_time and Average_Queue behaviors is quite complex and an in-depth analysis falls outside the goal

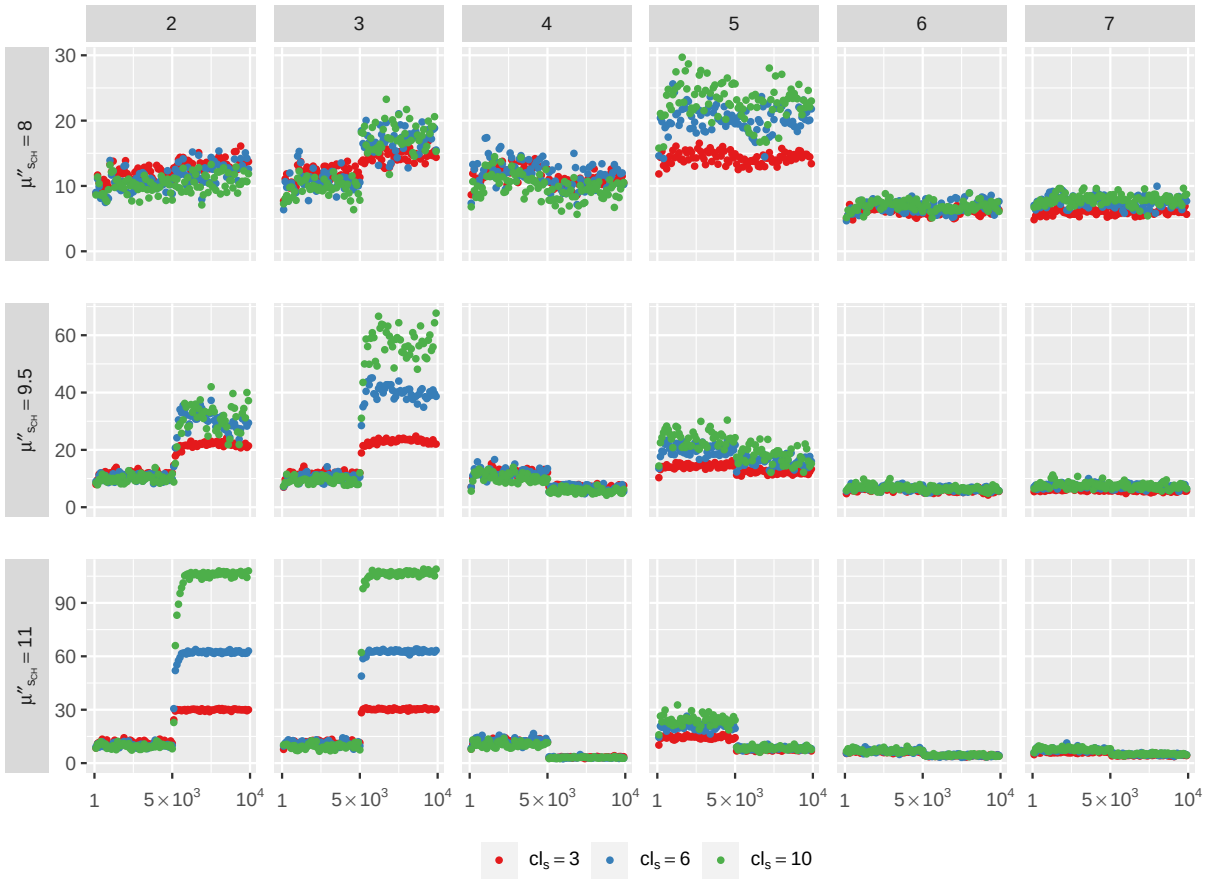


Fig. 6.16 Average Waiting time KPI behavior considering three different mean processing time increase levels $\mu_{s_{CH}}$ and three buffer capacity limits cl_s. Models with configuration $(s_{BN}, s_{CH}) = (5,3)$.

of this thesis, the observations here listed are always valid but insufficient to fully describe this dependence.

- When a resource loses production capacity, the higher the capacity limit cl_s is, the wider the increase of average Waiting_time in correspondence of the changing stage s_{CH}.
  This is visible in correspondence with the changing stage $s_{CH} = 3$, comparing models with different buffer capacity limits cl_s.

- When a resource loses production capacity but the system remains stable, the higher the capacity limit cl_s is, the less relevant the spread of average Waiting_time variation is in stages upstream and downstream the changing stage cl_s.
  This is visible upstream and downstream the changing stage $s_{CH} = 3$, comparing models

with different buffer capacity limits cl_s, having after-change mean processing time value $\mu''_{s_{CH}} = 8$ or $\mu''_{s_{CH}} = 9.5$.

To observe another impact of buffer capacity limits on Waiting_time, systems with unlimited buffer are briefly considered. Indeed, even if it is not a feasible situation, it can be interpreted as a case of systems having buffers with extremely high capacity limits. Figure 6.17 shows average

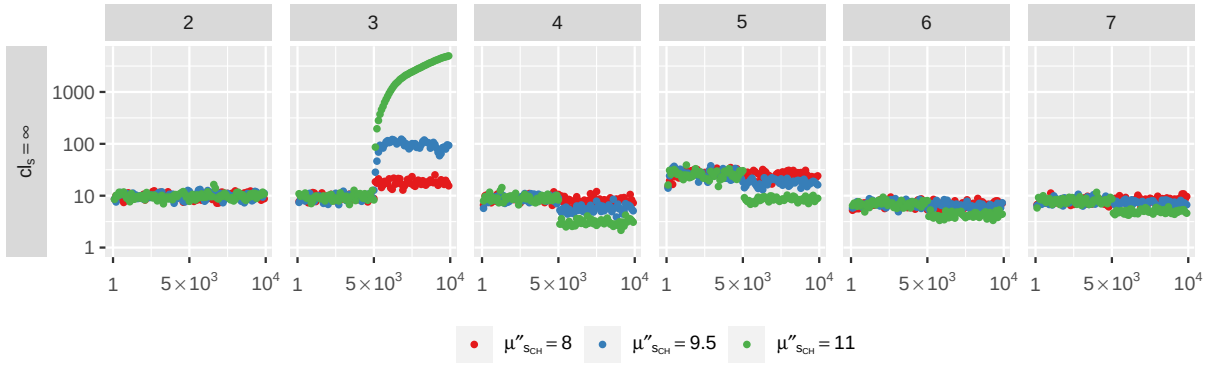


Fig. 6.17 Average Waiting time KPI behavior considering three different mean processing time increase levels $\mu_{s_{CH}}$ and buffers with unlimited capacity. Models with configuration $(s_{BN}, s_{CH}) = (5,3)$ and $cl_s = \infty$. Graphs with log-scale on y-axis.

Waiting_time behavior in case of a production capacity decrease when all buffers have unlimited capacity. The effects of high capacity buffers are taken to the extreme: when the changing stage s_{CH} processing time increases, average Waiting_time extensively decreases in stages downstream s_{CH} and increases in correspondence with the changing stage s_{CH}. If, after the variation, the system becomes unstable (as in model with $\mu''_{s_{CH}} = 11$), average Waiting_time of stage s_{CH} even grows towards infinity, since buffers have no boundaries and job mean inter-arrival in the first stage is lower than the mean processing time of the changing stage ($\mu_a < \mu''_{s_{CH}}$). However, it can be noticed that Waiting_time does not change in stage $s = 2$, upstream s_{CH}, even when the system becomes unstable.

The discussion about this particular (not realistic) case was included to show that buffers having high capacity limits work as temporary decoupling points, delaying variation effects on KPIs to spread upstream the stage in which they are positioned. The delay caused by infinite buffers (which are permanent decoupling points) is infinite, and so KPIs upstream never change.

Processing time decrease effects on Waiting_time and Average_Queue KPIs

The effects of a production capacity increase on Waiting_time and Average_Queue KPIs are the opposite than the ones caused by a production capacity decrease. Figure 6.18 shows Waiting_time

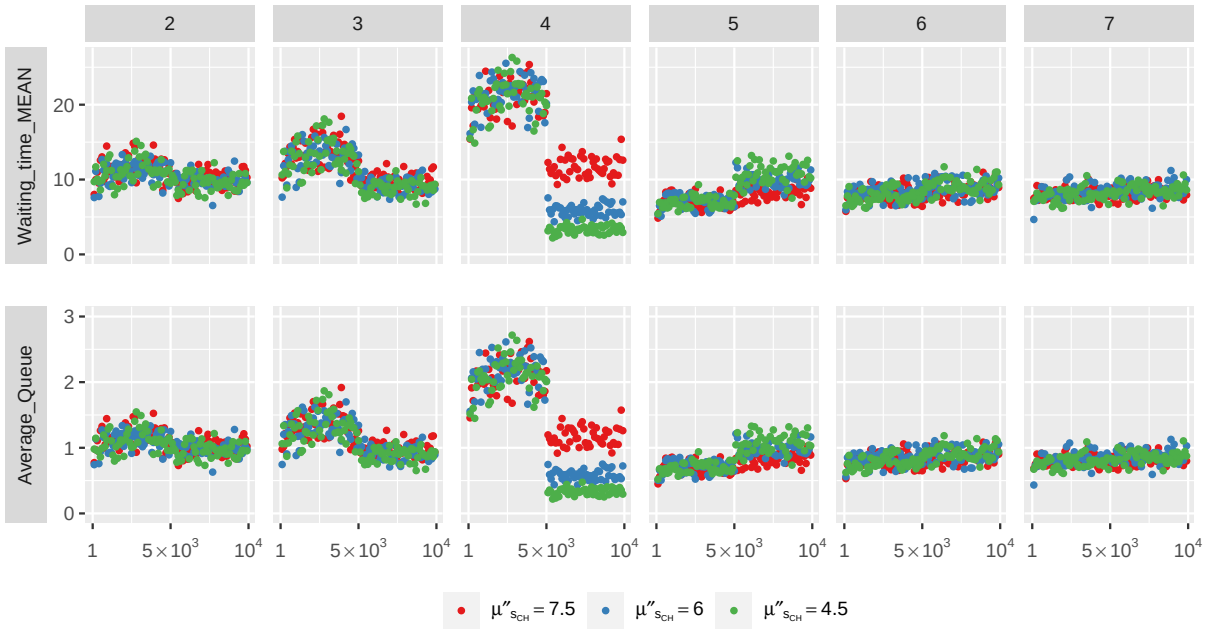


Fig. 6.18 Waiting time RW and Average Queue KPIs behaviors considering different processing time decrease levels $\mu''_{s_{CH}}$. Models with configuration $(s_{BN}, s_{CH}) = (4,4)$ and $cl_s = 6$.

and Average_Queue behaviors in case of a mean processing time $\mu_{s_{CH}}$ decrease.

- If a resource increases its production capacity
  - In stages upstream the changing stage s_{CH}, average Waiting_time and Average_Queue decrease.
  - In correspondence of the changing stage s_{CH}, average Waiting_time and Average_Queue decrease.
  - In stages downstream the changing stage s_{CH}, average Waiting_time and Average_Queue increase.
- Average Waiting_time and Average_Queue variations are wider the more significant is the increase of processing time.
- Average Waiting_time and Average_Queue variations in a certain stage s are less and less relevant the further stage s is from the changing stage s_{CH}, both upstream and downstream.

Waiting_time and Average_Queue KPI behaviors in the first stage of the line

It should be noted that the first stage has not been included in any figure of this subsection. 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 are present at the same time is not useful. Moreover, Waiting_time and Average_Queue of the first stage 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. Figure 6.19 shows the behaviors of Waiting_time and Average_Queue when

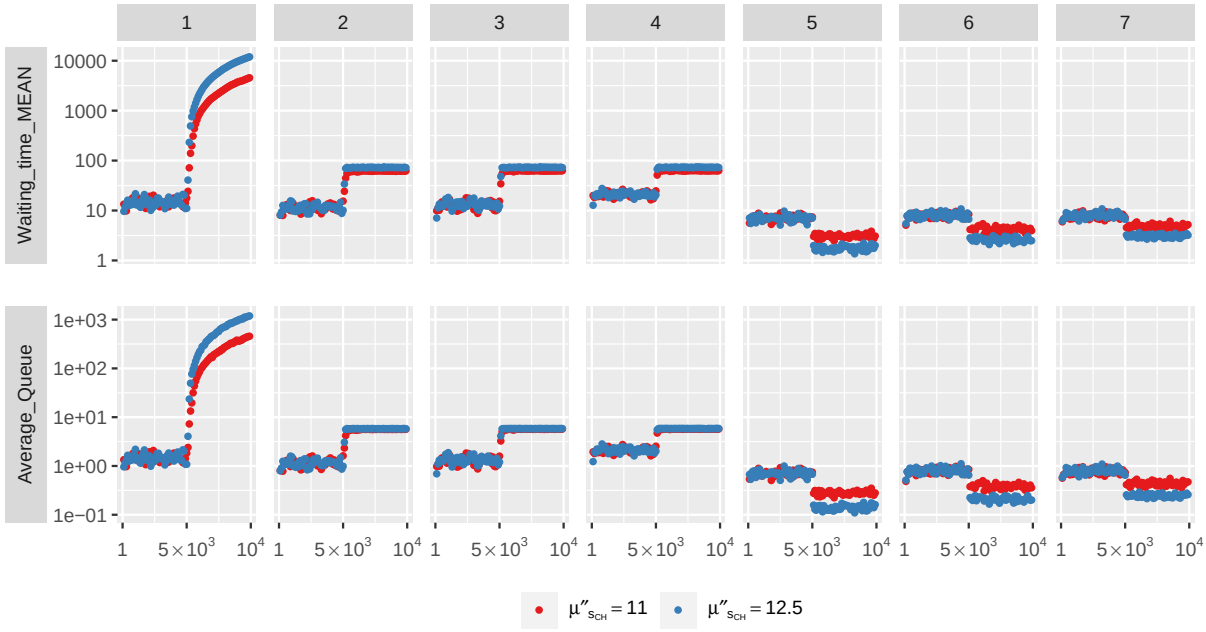


Fig. 6.19 Waiting time RW and Average Queue KPIs behaviors considering different processing time increase levels that make the system unstable. Models with configuration $(s_{BN}, s_{CH}) = (4,4)$ and $cl_s = 6$. Graphs with log-scale on y-axis.

the system becomes unstable: in the first stage both these KPIs increase towards infinity. In particular it should be noticed that, at the end of the process, but Average_Queue grows as if new jobs kept arriving, even if no more jobs enter the line and the buffer starts emptying. This error verifies because in this situation Waiting_time (rightly) keeps increasing, 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 been omitted in previous figures.

6.2 Buffer capacity variation

A buffer capacity variation does not have effects on resource processing time and, so, never influences the system cycle time. Therefore, this type of variation is visible only in KPIs related to buffers (i.e. Waiting_time, Blocking_time and Starving_time, and the relative Stage State KPIs), while the others (CCI KPIs, Processing_time and Utilization) are not subject to any change. Plots of KPIs not influenced by buffer capacity variations are not shown and discussed, to restrain the analysis to the most relevant indicators.

6.2.1 Waiting_time KPI

Waiting_time and Average_Queue are strongly influenced by buffer capacity limits (as shown in subsection 6.1.4) and are directly related to queue statuses. In this subsection Waiting_time behavior is analyzed in situations where a buffer capacity variation verifies, while Average_Queue is not discussed, since it shows the same behavior of Waiting_time. At the end of the subsection the problem of buffers having high capacity limits before the capacity variation is presented.

Buffer capacity increase effects on Waiting_time KPI

Figure 6.20 shows average Waiting_time behavior when a buffer capacity increases.

- If a buffer capacity increases
  - In stages upstream the changing stage s_{CH}, average Waiting_time decreases.
  - In correspondence of the changing stage s_{CH}, average Waiting_time increases.
  - In stages downstream the changing stage s_{CH}, average Waiting_time does not significantly change.
- Average Waiting_time variations are wider the larger the increase of buffer capacity is. This is visible comparing models with different after-change buffer capacities $cl''_{s_{CH}}$.
- Average Waiting_time variations are
  - wider if the changing stage is upstream or in correspondence with the bottleneck ($s_{CH} \leqslant s_{BN}$)
  - narrower if the changing stage is downstream the bottleneck ($s_{CH} > s_{BN}$)

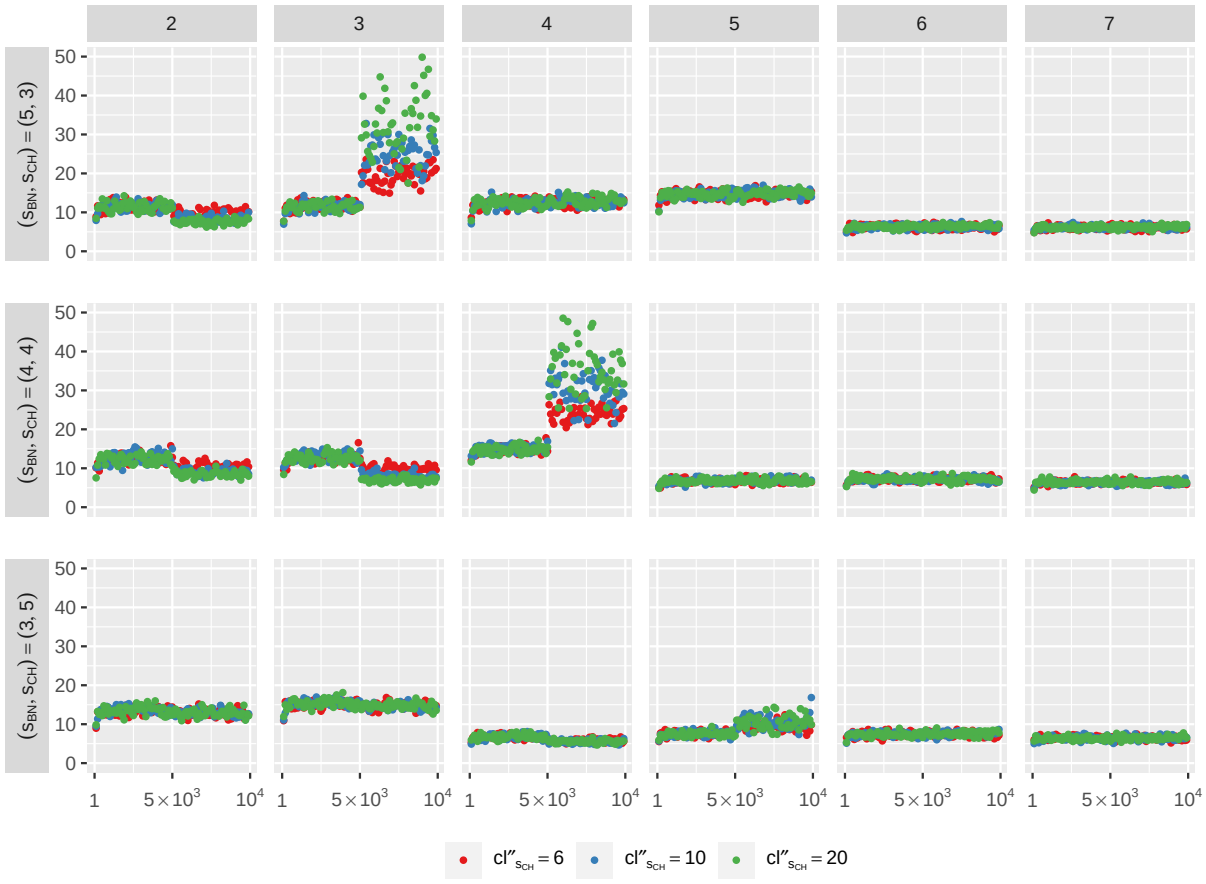


Fig. 6.20 Waiting time RW KPI behavior considering different buffer capacity increase levels $cl''_{s_{CH}}$ and three bottleneck - changing stage configurations (s_{BN}, s_{CH}). Models with $cl'_s = 3$.

This is visible comparing models having bottleneck-changing stage configurations $(s_{BN}, s_{CH}) = (5,3)$ (changing stage upstream the bottleneck) or $(s_{BN}, s_{CH}) = (4,4)$ (changing stage in correspondence with the bottleneck) with models having $(s_{BN}, s_{CH}) = (3,5)$ (changing stage downstream the bottleneck) .

- Average Waiting_time variations in a certain stage s are less and less relevant the further stage s is from the changing stage s_{CH}, both upstream and downstream s_{CH}.

Waiting_time behavior considering different initial buffer sizes cl'_s

In certain conditions, buffer capacity variations may have no effects on Waiting_time. The occurrence of a change of this KPI mainly relies on the initial buffer saturation levels before the capacity variation verifies. Figure 6.21 shows average Waiting_time behavior considering

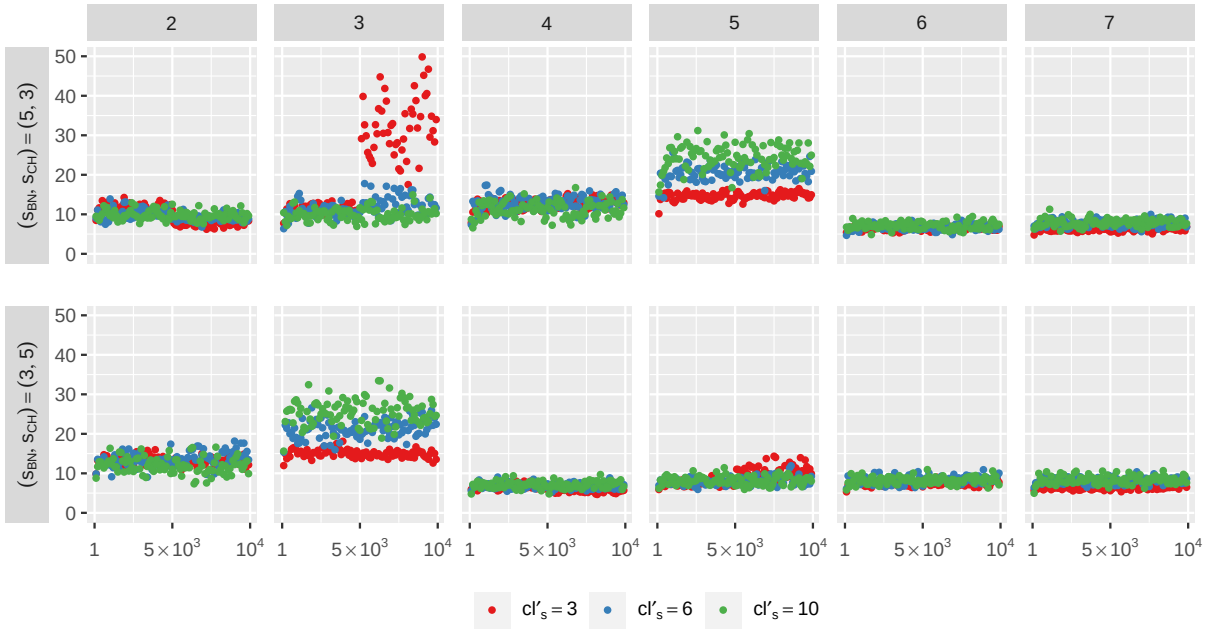


Fig. 6.21 Waiting time RW KPI behavior considering different initial buffer capacity levels cl'_s and two bottleneck-changing stage configurations (s_{BN}, s_{CH}). Models with $cl''_s = 20$.

different initial buffer capacities cl'_s.
When a capacity limit increases

- If buffers tend to be saturated, average Waiting_time significantly changes.
  This is visible in models with before-change buffer capacity $cl'_s = 3$, in particular in correspondence with the changing stage s_{CH}.
- If buffers tend to be unsaturated, average Waiting_time does not change.
  This is visible in models with before-change buffer capacity $cl'_s = 10$.
- If buffers are on average halfway saturated, average Waiting_time behavior mostly depends on the position of the changing stage with respect to the bottleneck.
  This is visible in models with before-change buffer capacity $cl'_s = 6$: when s_{CH} is upstream s_{BN} (i.e. model with configuration $(s_{BN}, s_{CH}) = (5,3)$), average Waiting_time (slightly) changes in correspondence of s_{CH}.[6] When s_{CH} is downstream s_{BN} (i.e. model with configuration $(s_{BN}, s_{CH}) = (3,5)$), average Waiting_time does not significantly change of s_{CH}

[6]This also verifies when the variation occurs in correspondence with the changing stage s_{CH}

Buffer capacity decrease effects on Waiting_time KPI

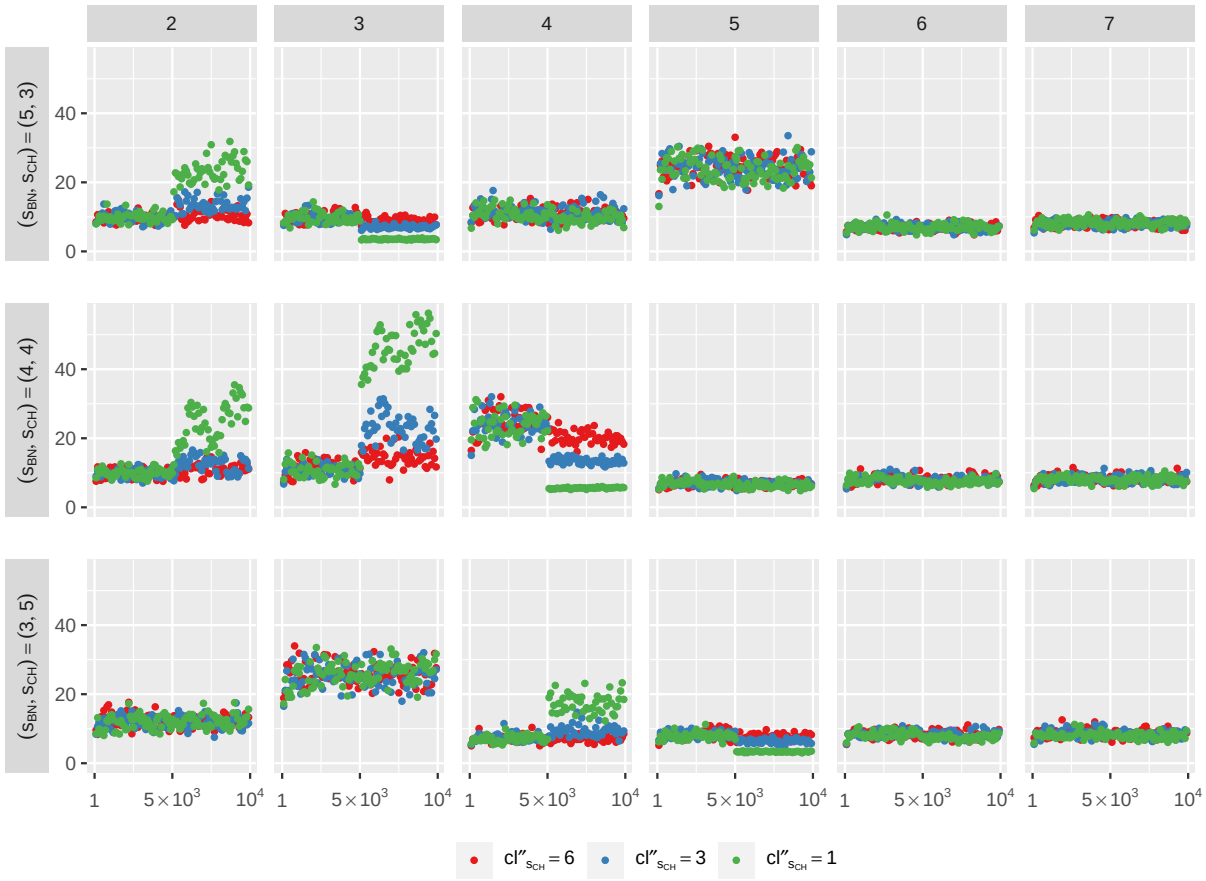


Fig. 6.22 Waiting time RW KPI behavior considering different buffer capacity decrease levels cl''_s. Models with $cl'_s = 10$)

Figure 6.22 shows average Waiting_time behavior when a buffer capacity decreases.

- If a buffer capacity reduces
  - In stages upstream the changing stage s_{CH}, average Waiting_time increases.
  - In correspondence of the changing stage s_{CH}, average Waiting_time decreases.
  - In stages downstream the changing stage s_{CH}, average Waiting_time does not significantly change.
- Average Waiting_time variations are wider the larger the decrease of buffer capacity is. This is visible comparing models with different after-change buffer capacities $cl''_{s_{CH}}$.

- Average Waiting_time variations are
  - wider if the changing stage is upstream or in correspondence with the bottleneck ($s_{CH} \leqslant s_{BN}$)
  - narrower if the changing stage is downstream the bottleneck ($s_{CH} > s_{BN}$)

  This is visible comparing models having bottleneck-changing stage configurations $(s_{BN}, s_{CH}) = (5,3)$ (changing stage upstream the bottleneck) or $(s_{BN}, s_{CH}) = (4,4)$ (changing stage in correspondence with the bottleneck) with models having $(s_{BN}, s_{CH}) = (3,5)$ (changing stage downstream the bottleneck).
- Average Waiting_time variations in a certain stage s are less and less relevant the further stage s is from the changing stage s_{CH}, both upstream and downstream.

Variation conceal caused by before-change high capacity buffers

It is important to highlight that not every buffer capacity limit variation is detectable using Waiting_time or Average_Queue. Indeed, when before-change buffer sizes are much higher than necessary, and so underused and largely unsaturated during the process, buffer capacity growths or limited capacity reductions are impossible to notice. It verifies because, if queues in buffers are consistently much lower than the capacity limit, a variation of the limit will not have effects on the queue: in case of a capacity increase, the added room is not used, while in case of a capacity decrease (if not too large), the removed space was not used before the change and, so, jobs are not influenced by the change. Figure 6.23 helps to visualize this kind of situation: all the

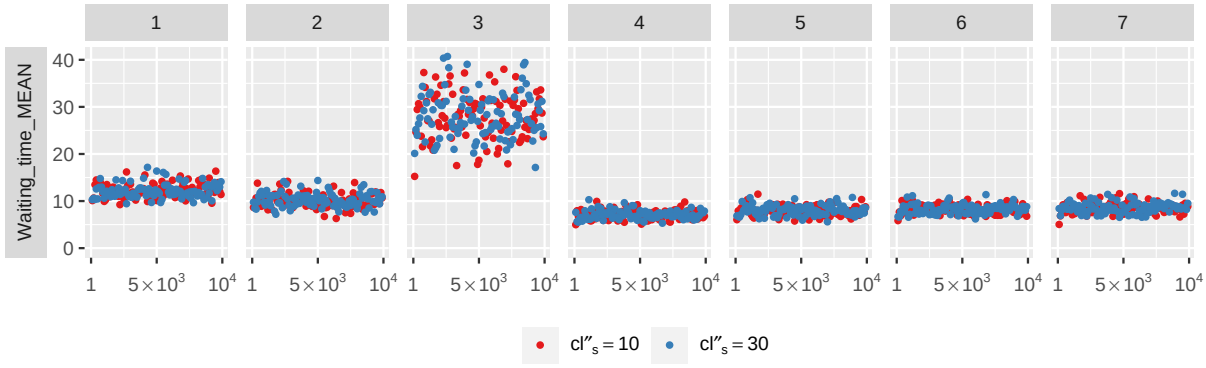


Fig. 6.23 Waiting time RW KPI behavior considering different buffer capacity variation levels cl''_s. Models with configuration $(s_{BN}, s_{CH}) = (3,5)$ and $cl'_s = 20$.

buffers have high capacities ($cl'_s = 20$), and so are extremely unsaturated. When the changing

stage capacity limit increases to $cl_{s_5}s = 30$, average Waiting_time behavior does not change. Moreover, since the changing stage is downstream the bottleneck ($s_{CH} = 5 > s_{BN} = 3$), even a capacity limit decrease has no visible effects on average Waiting_time.
Before-change high capacity buffers have also the same effects on Blocking_time and Starving_time KPIs, which are discussed in the next chapter. Therefore, it is possible to conclude that, if a buffer capacity change occurs in correspondence with a high capacity and extremely unsaturated buffer, it is not detectable with any KPI defined in this thesis.

6.2.2 Blocking_time and Starving_time KPIs

Since Blocking_time and Starving_time KPI behaviors are specular (as seen in subsection 6.1.3), they are analyzed together in this subsection. These KPI behaviors are influenced by both the production capacities of resources and the capacities of buffers.

Buffer capacity increase effects on Blocking_time and Starving_time KPIs

Fig. 6.24 Blocking time RW KPI behavior considering different buffer capacity increase levels $cl''_{s_{CH}}$ and three bottleneck - changing stage configurations (s_{BN}, s_{CH}). Models with $cl'_s = 3$.

Figure 6.24 and figure 6.25 show, respectively, average Blocking_time and average Starving_time behaviors in case of buffer capacity increases.

- If a buffer capacity increases

Fig. 6.25 Starving time RW KPI behavior considering different buffer capacity increase levels $cl''_{s_{CH}}$ and three bottleneck - changing stage configurations (s_{BN}, s_{CH}). Models with $cl'_s = 3$.

  - In stages upstream the changing stage s_{CH}, average Blocking_time decreases and average Starving_time increases.
  - In correspondence of the changing stage s_{CH}, average Blocking_time increases and average Starving_time decreases.
  - In stages downstream the changing stage s_{CH}, both average Blocking_time and average Starving_time do not significantly change.

- Average Blocking_time and average Starving_time variations are wider the larger the increase of buffer capacity is.
  This is visible comparing models with different after-change buffer capacities $cl''_{s_{CH}}$.

- Average Blocking_time and average Starving_time variations are

- wider if the changing stage is upstream or in correspondence with the bottleneck ($s_{CH} \leqslant s_{BN}$)
- narrower if the changing stage is downstream the bottleneck ($s_{CH} > s_{BN}$)

This is visible comparing models having bottleneck-changing stage configurations $(s_{BN}, s_{CH}) = (5,3)$ (changing stage upstream the bottleneck) or $(s_{BN}, s_{CH}) = (4,4)$ (changing stage in correspondence with the bottleneck) with models having $(s_{BN}, s_{CH}) = (3,5)$ (changing stage downstream the bottleneck) .

- Average Blocking_time and average Starving_time variations in a certain stage s are less and less relevant the further stage s is from the changing stage s_{CH}, both upstream and downstream s_{CH}.

Blocking_time and Starving_time behaviors considering different initial buffer sizes cl'_s

As for Waiting_time, buffer capacity variations may have no effects on Blocking_time and Starving_time. The occurrence of a change of these KPIs mainly rely on the initial buffer saturation levels before the capacity variation verifies. Figure 6.21 shows average Blocking_time

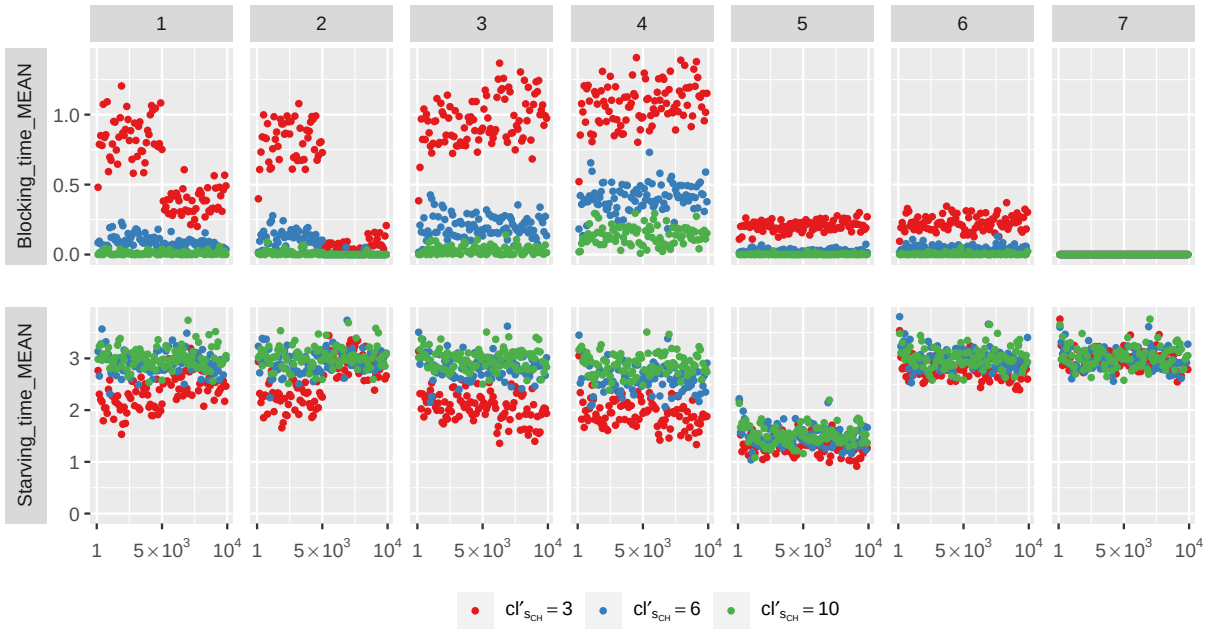


Fig. 6.26 Blocking time and Starving time RW KPIs behaviors considering different initial buffer capacity levels cl'_s. Models with configuration $(s_{BN}, s_{CH}) = (5,3)$ and after-change value $cl''_s = 20$.

and Starving_time behaviors considering different initial buffer capacities cl'_s.
When a capacity limit increases:

- If buffers tend to be saturated, average Blocking_time and Starving_time significantly change.
  This is visible in models with before-change buffer capacity $cl'_s = 3$, in particular in correspondence with the changing stage $s_{CH} = 3$.
- If buffers tend to be unsaturated, average Blocking_time and Starving_time do not change.
  This is visible in models with before-change buffer capacity $cl'_s = 10$.
- If buffers are on average halfway saturated, average Blocking_time and Starving_time behaviors mostly depend on the position of the changing stage with respect to the bottleneck.
  This is visible in models with before-change buffer capacity $cl'_s = 6$: when s_{CH} is upstream s_{BN} (i.e. model with configuration $(s_{BN}, s_{CH}) = (5,3)$), average Blocking_time and Starving_time (slightly) change in correspondence and upstream s_{CH}. When s_{CH} is downstream s_{BN} (i.e. model with configuration $(s_{BN}, s_{CH}) = (3,5)$), average Blocking_time and Starving_time do not significantly change.

Buffer capacity decrease effects on Blocking_time and Starving_time

Figure 6.22 shows average Blocking_time and Starving_time behaviors when a buffer capacity decreases.

- If a buffer capacity reduces
  - In stages upstream the changing stage s_{CH}, average Blocking_time increases and average Starving_time decreases.
  - In correspondence of the changing stage s_{CH}, average Blocking_time decreases and average Starving_time increases.
  - In stages downstream the changing stage s_{CH}, both average Blocking_time and average Starving_time do not significantly change.
- Average Blocking_time and Starving_time variations are wider the larger the decrease of buffer capacity is.
  This is visible comparing models with different after-change buffer capacities $cl''_{s_{CH}}$.
- Average Blocking_time and Starving_time variations are

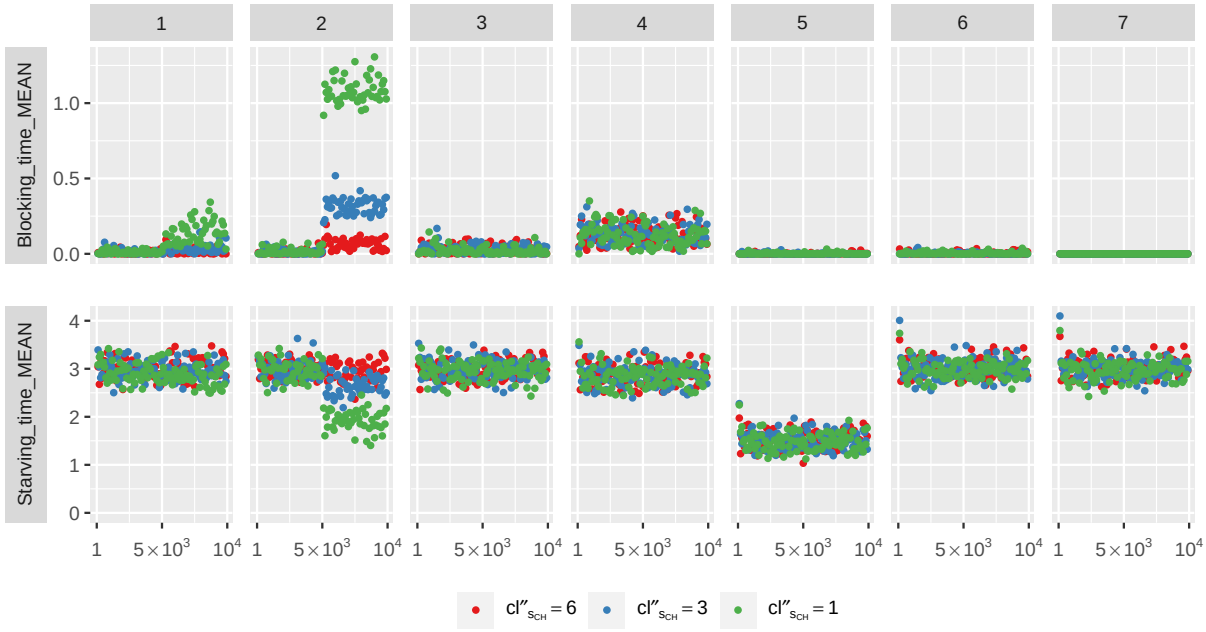


Fig. 6.27 Blocking time and Starving time RW KPIs behaviors considering different buffer capacity decrease levels cl''_s. Models with configuration $(s_{BN}, s_{CH}) = (5,3)$ and $cl'_s = 10$.

  - wider if the changing stage is upstream or in correspondence with the bottleneck ($s_{CH} \leqslant s_{BN}$)
  - narrower if the changing stage is downstream the bottleneck ($s_{CH} > s_{BN}$)

- Average Blocking_time and Starving_time variations in a certain stage s are less and less relevant the further stage s is from the changing stage s_{CH}, both upstream and downstream.

Chapter 7

A map of variations

This chapter provides a summary of variations, based on the results analyzed in chapter 6. The summary aims to fulfill the thesis goal outlined in section 2.1.1, which is to define a map of variations to serve as a handbook that allows to interpret the indicator behaviors and relate a KPI change with a variation occurring in the real system.
The first section is dedicated to change detection, that is the step preliminary to the change identification (see introduction of chapter 2), in order to specify what system variation is possible to notice monitoring a certain KPI.
In the second section, the process variation effects on each indicator is briefly described and visualized through schemes representing abstractions of the KPI average behaviors, in order to outline if and how far they are fitting to give insights about the system variation extents.

7.1 Variation detection

The goal of this section is to answer to two questions

Which KPIs are able to detect a certain process variation?

In what situation can a KPI detect a certain process variation?

Table 7.1 Processing time variation detection

KPI	Processing time		Detection condition
	Increase ⇑	Decrease ⇓	
Input_diff	✓		System becomes unstable
Mid_diff	✓		System becomes unstable
Output_diff	✓		System becomes unstable
Waiting_time	✓	✓	
Average_Queue	✓	✓	
Processing_time	✓	✓	
Utilization	✓	✓	
Blocking_time	✓	✓	
Blocking_prob	✓	✓	
Starving_time	✓	✓	
Starving_prob	✓	✓	

Table 7.2 Buffer capacity variation detection

KPI	Buffer capacity		Detection condition
	Increase ⇑	Decrease ⇓	
Input_diff			
Mid_diff			
Output_diff			
Waiting_time	✓	✓	Buffer not highly unsaturated
Average_Queue	✓	✓	Buffer not highly unsaturated
Processing_time			
Utilization			
Blocking_time	✓	✓	Buffer not highly unsaturated
Blocking_prob	✓	✓	Buffer not highly unsaturated
Starving_time	✓	✓	Buffer not highly unsaturated
Starving_prob	✓	✓	Buffer not highly unsaturated

Tables 7.1 and 7.2 defines which KPIs and under what condition are able to detect the different process variations. They can be further summarized as follows

- Consecutive Cases Intervals KPIs (Input_diff, Mid_diff, and Output_diff) change only if the system becomes unstable.
- Processing_time and Utilization KPIs change only when the system variation involves resource production capacities (i.e. processing time variations).
- Waiting_time, Blocking_time, Starving_time, and relative Stage State KPIs (Average_Queue, Blocking_prob, and Starving_prob) change when both processing time or buffer capacity variations occur, but the second type of variation affects these KPIs only if the buffers (in particular the one in correspondence of the changing stage) are not too underused.

7.2 Variation identification

As in the previous section, also this one aims to answer to two questions

Is a variation recognizable and distinguishable looking at how it impacts on the considered KPIs?

Is the variation extent clearly identifiable looking at how it impacts on the considered KPIs?

Therefore, this section focuses on summarizing the system variation effects on KPIs.
For each indicator, a description of its behavior and a brief comment on its usefulness are provided; then, six figures are attached, to help to visualize the KPI average behavior when a variation (increase or decrease) occurs, looking at stages upstream, in correspondence, and downstream the changing stage.

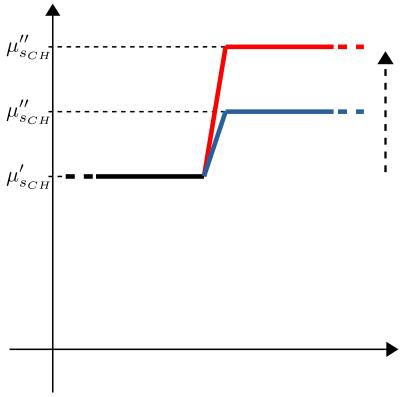


Fig. 7.1 Example of scheme displaying the variation effects on a KPI in a certain stage

Figure 7.1 is an example plot, showing the behavior of Processing_time KPI in correspondence of the changing stage when a processing time increase occurs.

- The black line represent the average KPI behavior **before** the variation occurs
- The blue line represent the average KPI behavior **after** the variation has occurred, in case the system remained **stable**
- The red line represent the average KPI behavior **after** the variation has occurred, in case the system became **unstable**

In these schemes, when one or more parameters are present on the y-axis (e.g. $\mu'_{s_{CH}}$ and $\mu''_{s_{CH}}$ in the example), they represent the value which the KPI settles around before or after the variation. If no parameters are present, the lines just show the approximate KPI behavior.
Moreover, in schemes, dashed arrows are used to visualize variations. Note that the arrow lengths are not exactly proportionate to actual KPI changes, but they only help to show the direction and the approximate extent of the indicator variations.
To simplify the dissertation, since their behaviors are very similar, KPIs and respective Stage State KPIs are discussed together.

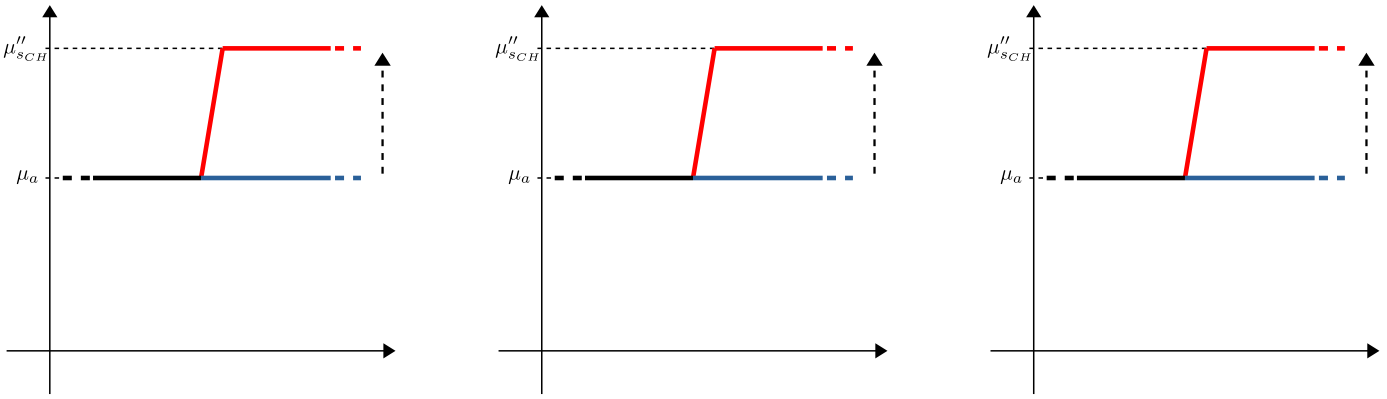
7.2.1 Processing time variation

Effects on Consecutive Cases Intervals KPIs

Input_diff, Mid_diff, and Output_diff are discussed together, since their behaviors are mostly similar. CCI KPIs behave as follow:

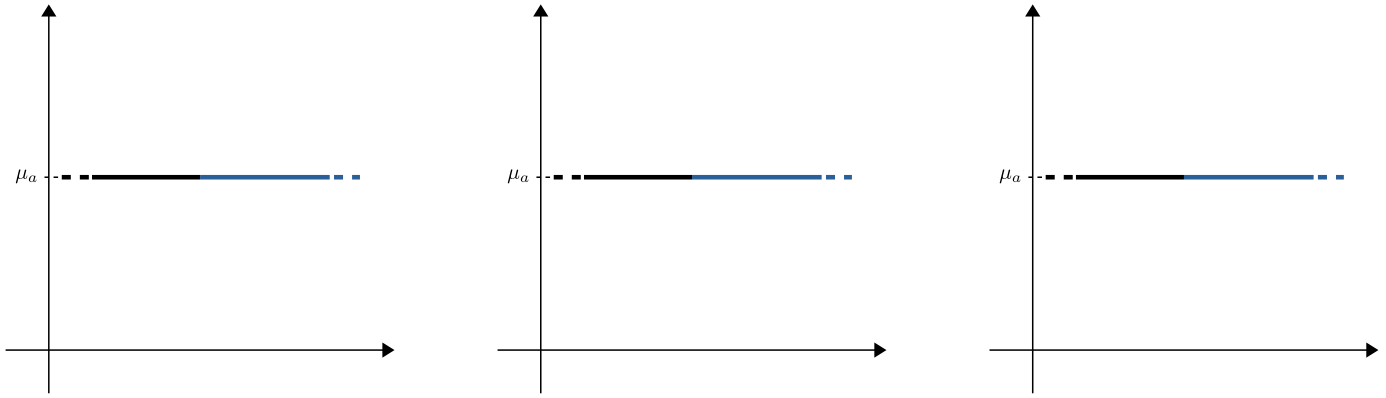
- If a processing time increase occurs ($\mu''_{s_{CH}} > \mu'_{s_{CH}}$)
  - If the system remains stable ($\mu''_{s_{CH}} < \mu_a$), CCI KPIs do not change, keeping settled around the average inter-arrival time μ_a.
  - If the system becomes unstable ($\mu''_{s_{CH}} > \mu_a$), CCI KPIs increase equally in all stages, settling around the after-change mean processing time of the changing stage $\mu''_{s_{CH}}$.
- If a processing time decrease occurs ($\mu''_{s_{CH}} < \mu'_{s_{CH}}$): CCI KPIs do not change, keeping settled around the average inter-arrival time μ_a.

CCI KPIs usefulness CCI KPIs are reliable indicators of the system cycle time and, therefore, can be used to understand if the system is still stable after a variation. 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.
Lastly, it must be added that Input_diff of the first stage has a behavior that differs from CCI KPIs computed in any other position: since the first stage buffer is unlimited, it is always an inter-arrival time indicator, rather than a cycle time indicator.



(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.2 Processing time increase effects on Consecutive Cases Intervals KPIs



(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.3 Processing time decrease effects on Consecutive Cases Intervals KPIs

Effects on Processing_time and Utilization KPIs

Processing_time KPI behaves as follows:

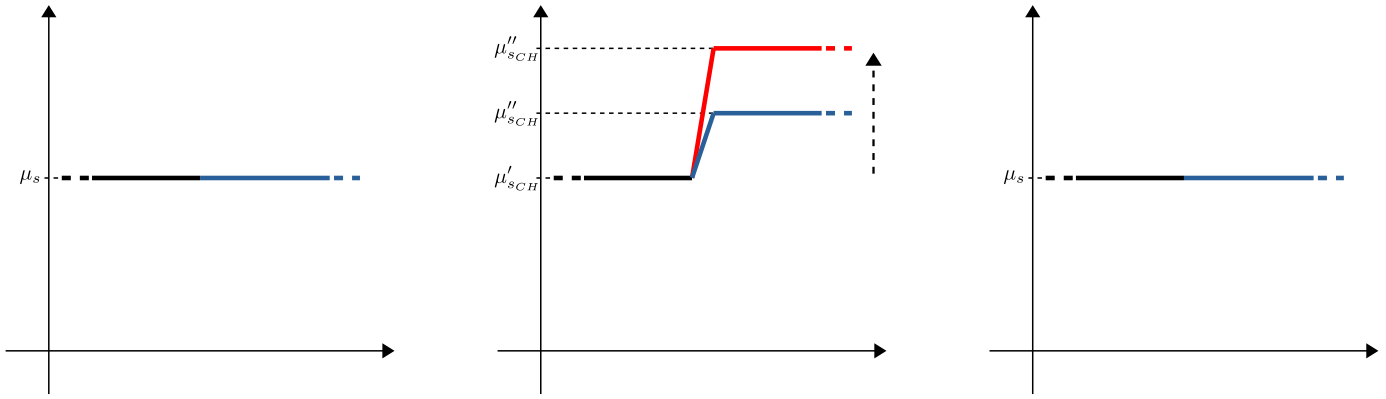
- When a processing time increase occurs ($\mu''_{s_{CH}} > \mu'_{s_{CH}}$)
  - In the changing stage, Processing_time increases, keeping aligned with $\mu''_{s_{CH}}$.
  - In other stages, Processing_time does not change.
- When a processing time decrease occurs ($\mu''_{s_{CH}} < \mu'_{s_{CH}}$)
  - In the changing stage, Processing_time decreases, keeping aligned with $\mu''_{s_{CH}}$.
  - In other stages, Processing_time does not change.

Utilization KPI behaves as follows:

- When a processing time increase occurs ($\mu''_{s_{CH}} > \mu'_{s_{CH}}$)
  - If the system remains stable ($\mu''_{s_{CH}} < \mu_a$)
    - In the changing stage, Utilization increases.
    - In other stages, Utilization does not change.
  - If the system becomes unstable ($\mu''_{s_{CH}} > \mu_a$)
    - In the changing stage, Utilization increases.
    - In other stages, Utilization decreases.
- When a processing time decrease occurs ($\mu''_{s_{CH}} < \mu'_{s_{CH}}$)
  - In the changing stage, Utilization decreases.
  - In other stages, Utilization does not change.

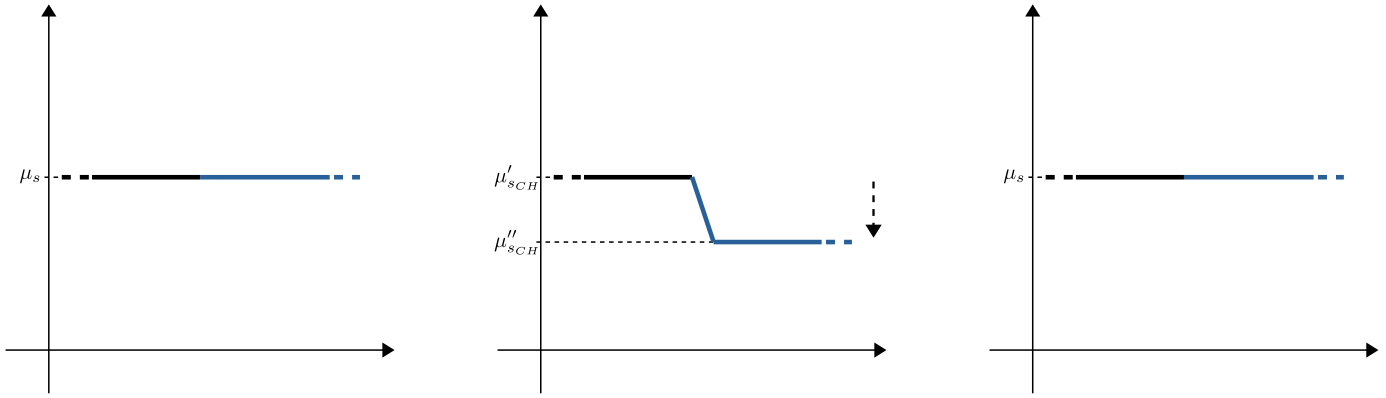
The variation extent of Utilization is wider the larger the processing time variation is. Values taken by Utilization are limited to the range $[0, 1]$.

Processing_time and Utilization KPIs usefulness Both these KPIs give clear information regarding the resources: Processing_time tells the absolute value of the average time needed by a resource to process a job, while Utilization shows how much a resource is exploited, comparing its processing time to the system cycle time.
Processing time variations directly impact on these KPIs, which turn out to be the most informative and swift-reacting among the considered indicators when this kind of change occurs.



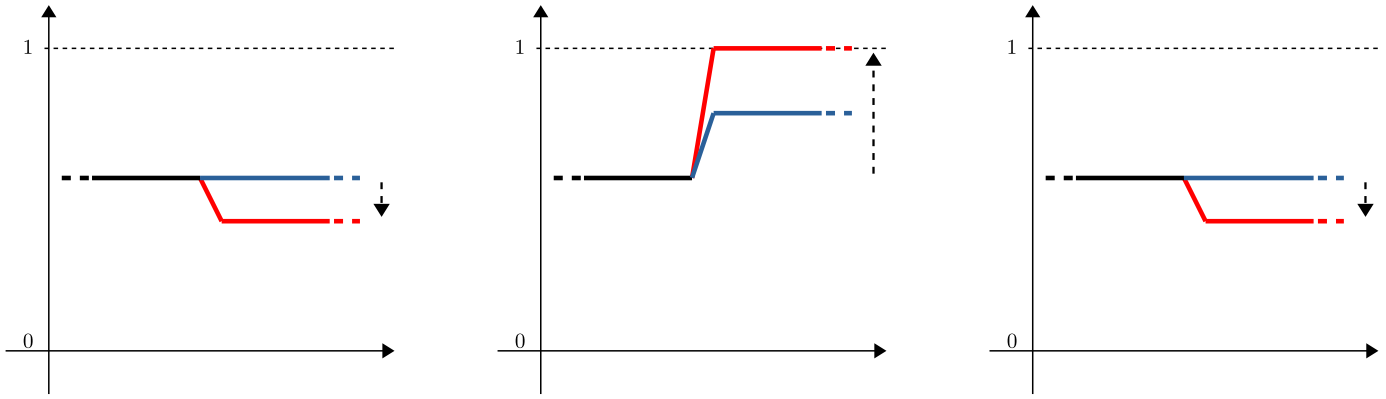
(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.4 Processing time increase effects on Processing time KPI



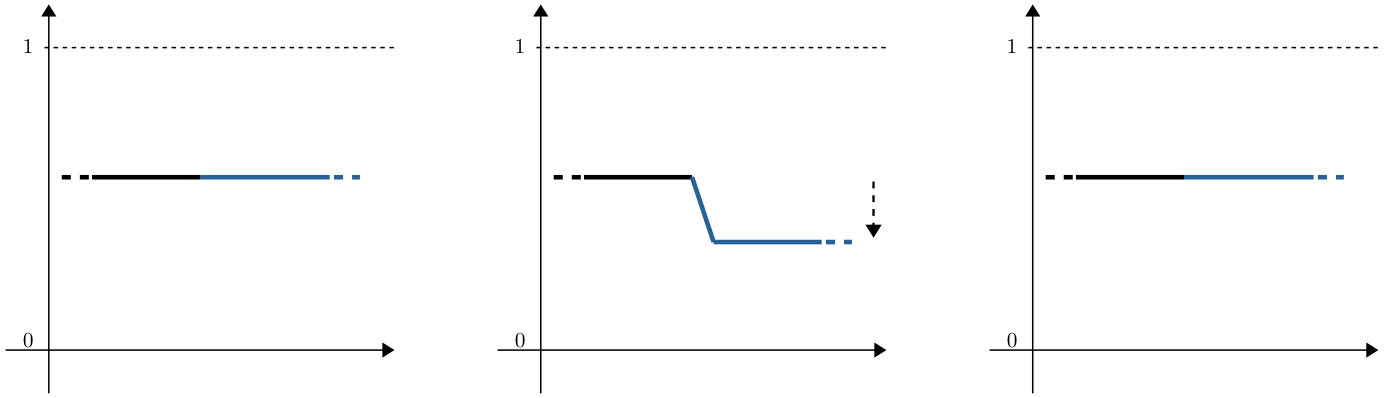
(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.5 Processing time decrease effects on Processing time KPI



(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.6 Processing time increase effects on Utilization KPI



(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.7 Processing time decrease effects on Utilization KPI

Effects on Waiting_time and Average_Queue KPIs

Waiting_time and Average_Queue KPIs behave as follows:

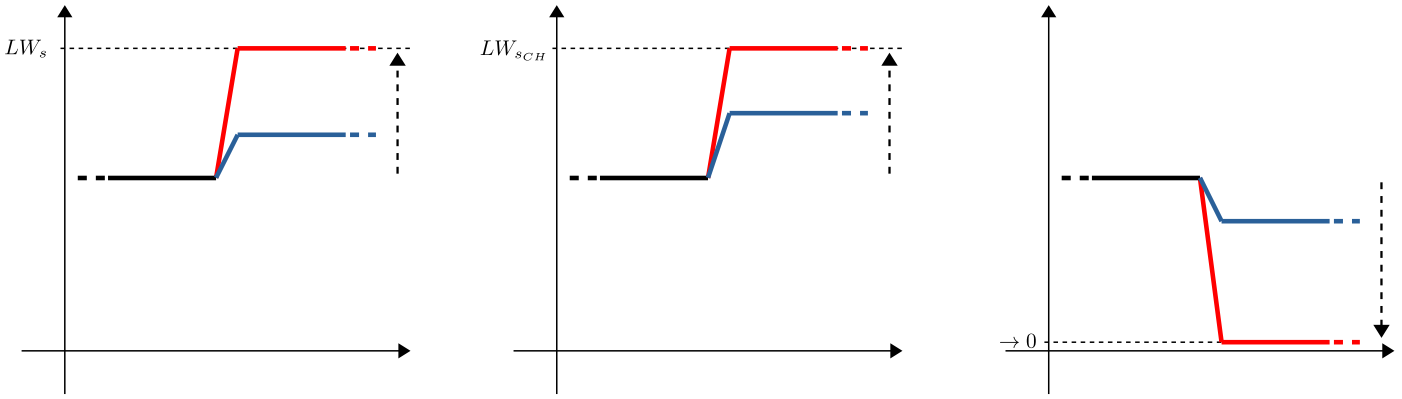
- When a processing time increase occurs ($\mu''_{s_{CH}} > \mu'_{s_{CH}}$)
  - If the system remains stable ($\mu''_{s_{CH}} < \mu_a$)
    - In stages upstream and in correspondence with the changing stage, Waiting_time and Average_Queue increase.
    - In stages downstream the changing stage, Waiting_time and Average_Queue decrease.
  - If the system becomes unstable ($\mu''_{s_{CH}} > \mu_a$)
    - In stages upstream and in correspondence with the changing stage
      - Waiting_time increases and settles around LW_s [1].
      - Average_Queue increases and settles around the buffer capacity limit of stage s, cl_s.
    - In stages downstream the changing stage, Waiting_time and Average_Queue decrease towards 0.
- When a processing time decrease occurs ($\mu''_{s_{CH}} < \mu'_{s_{CH}}$)
  - In stages upstream and in correspondence with the changing stage, Waiting_time and Average_Queue decrease.
  - In stages downstream the changing stage, Waiting_time and Average_Queue increase.

The variation extent of Waiting_time and Average_Queue in a certain stage s is influenced by stage s position, the processing time variation width ($|\mu''_{s_{CH}} - \mu'_{s_{CH}}|$), and the buffer capacity limits cl_s. It is possible to say that the variation extent is

- wider the larger the processing time variation ($|\mu''_{s_{CH}} - \mu'_{s_{CH}}|$) is.
- wider the nearer stage s is with respect to the changing stage s_{CH}.
- wider in stages near the changing stage s_{CH}, the higher the buffer capacity limits cl_s are.
- narrower in stages far from the changing stage s_{CH}, the higher the buffer capacity limits cl_s are.

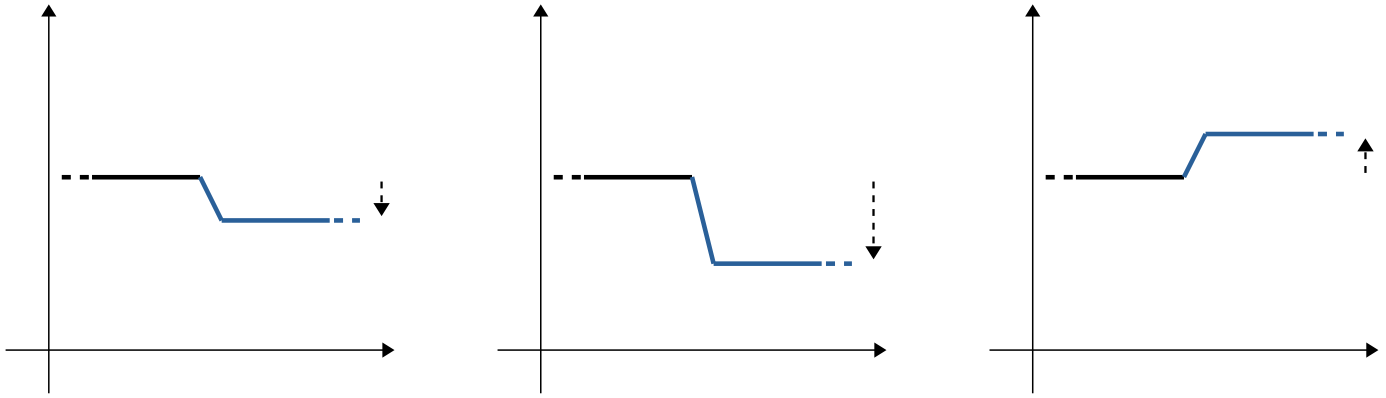
[1] LW_s is equal to the product of average cycle time (indicated by average Input_diff) and the buffer capacity limit of stage s, cl_s

Waiting_time and Average_Queue KPIs usefulness Waiting_time and Average_Queue offer two different perspectives regarding the buffers: Waiting_time has a job-oriented point of view, showing how much time jobs spent in buffers, while Average_Queue is more queue-oriented, allowing to know how many jobs are on average simultaneously present in a buffer.
These KPIs allow to understand a processing time variation direction (increase or decrease) and its approximate size: Waiting_time and Average_Queue do not show the exact change extent, but high KPI changes are the clue that a big system variation is occurring in a stage.



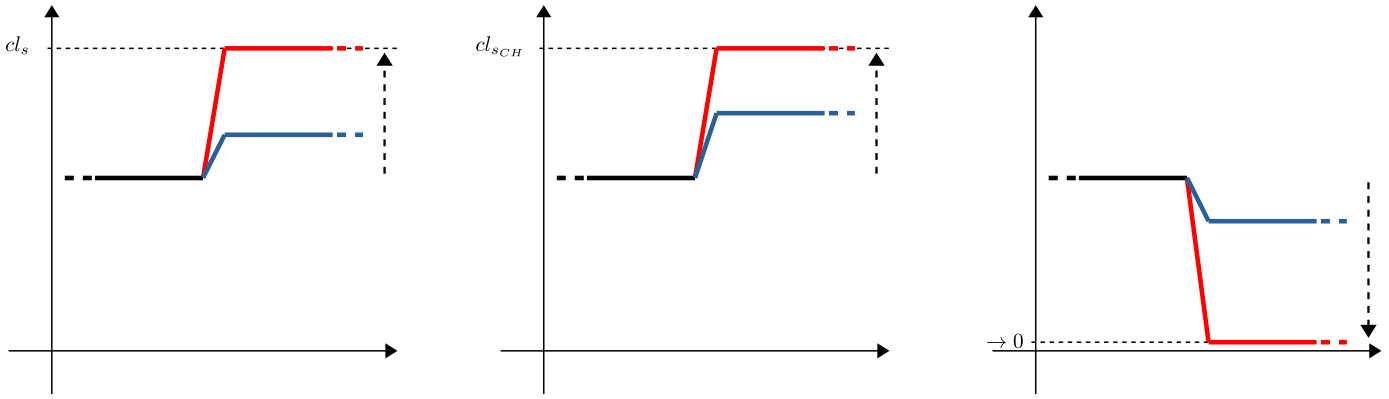
(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.8 Processing time increase effects on Waiting time KPI



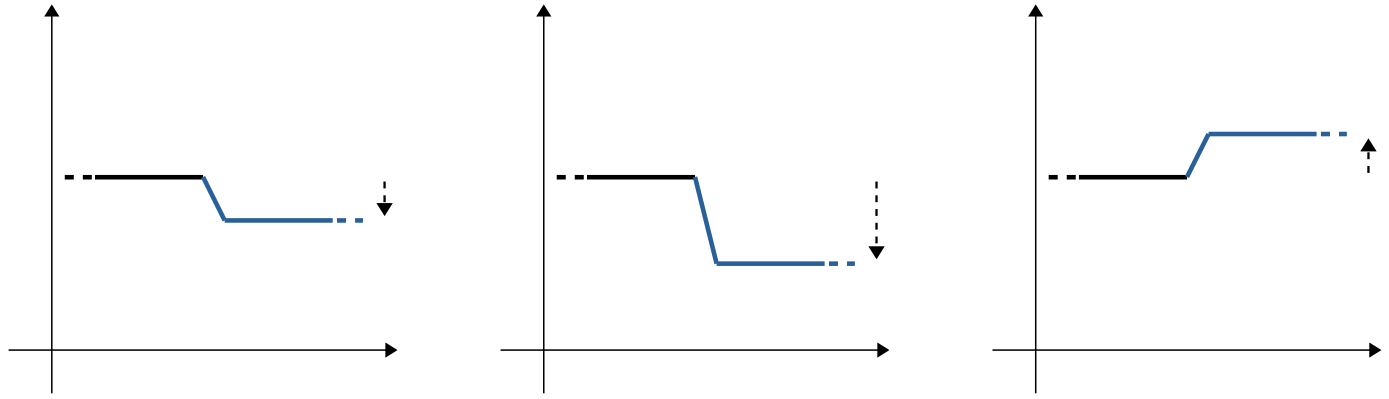
(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.9 Processing time decrease effects on Waiting time KPI



(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.10 Processing time increase effects on Average Queue KPI



(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.11 Processing time decrease effects on Average Queue KPI

Effects on Blocking_time and Blocking_prob KPIs

Blocking_time and Blocking_prob KPIs behave as follows:

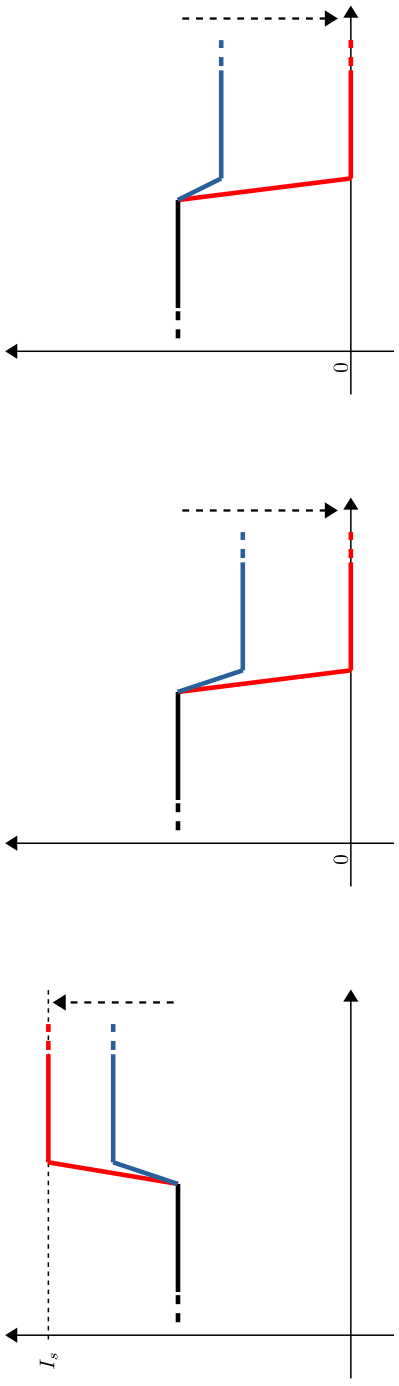
- When a processing time increase occurs ($\mu''_{s_{CH}} > \mu'_{s_{CH}}$)
  - If the system remains stable ($\mu''_{s_{CH}} < \mu_a$)
    - In stages upstream the changing stage, Blocking_time and Blocking_prob increase.
    - In stages downstream and in correspondence with the changing stage, Blocking_time and Blocking_prob decrease.
  - If the system becomes unstable ($\mu''_{s_{CH}} > \mu_a$)
    - In stages upstream the changing stage
      - Blocking_time increases and settles around I_s [2].
      - Blocking_prob increases.
    - In stages downstream and in correspondence with the changing stage, Blocking_time and Blocking_prob decrease and settle around 0.
- When a processing time decrease occurs ($\mu''_{s_{CH}} < \mu'_{s_{CH}}$)
  - In stages upstream the changing stage, Blocking_time and Blocking_prob decrease.
  - In stages downstream and in correspondence with the changing stage, Blocking_time and Blocking_prob increase.

The variation extent of Blocking_time and Blocking_prob in a certain stage s is influenced by stage s position, the processing time variation width ($|\mu''_{s_{CH}} - \mu'_{s_{CH}}|$), and the buffer capacity limits cl_s. It is possible to say that the variation extent is

- wider the larger the processing time variation ($|\mu''_{s_{CH}} - \mu'_{s_{CH}}|$) is
- wider the nearer stage s is with respect to the changing stage s_{CH}
- wider in stages near the changing stage s_{CH}, the higher the buffer capacity limits cl_s are
- narrower in stages far from the changing stage s_{CH}, the higher the buffer capacity limits cl_s are

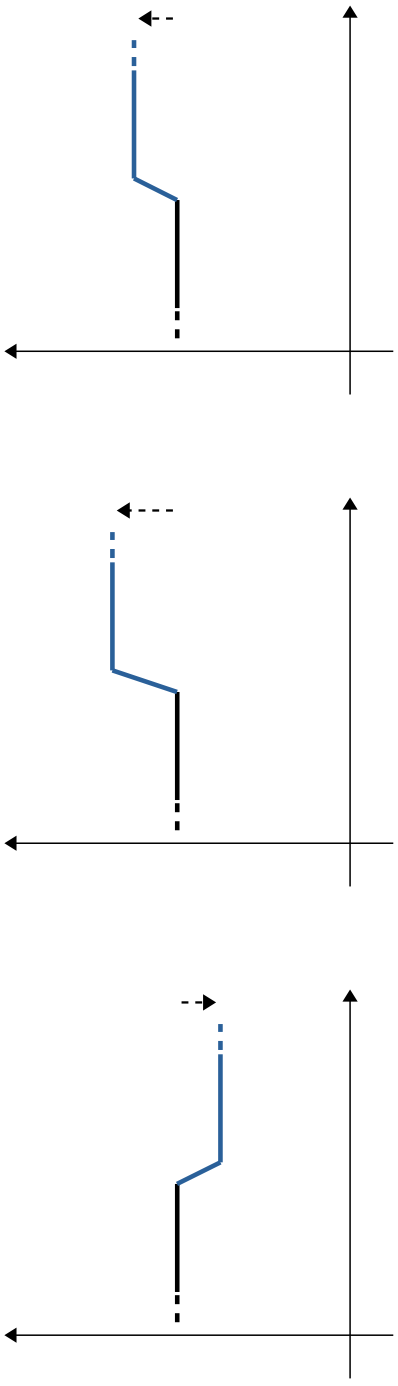
[2] I_s is equal to the difference between average cycle time (indicated by Mid_diff) and average processing time (indicated by Processing_time) of stage s

Blocking_time and Blocking_prob KPIs usefulness Blocking_time and Blocking_prob allow to understand how deeply stages different from the changing stage are affected by a processing time variation: Blocking_time shows how much time jobs spend seizing a resource after being processed, because the following buffer is full, while Blocking_prob shows how much of its available time a resource on average spends without working because a job could not leave it. These KPIs allow to understand a processing time variation direction (increase or decrease) and its approximate size: Blocking_time and Blocking_prob do not directly indicate the exact change extent, but high KPI changes are the clue that a big system variation is occurring in a stage.



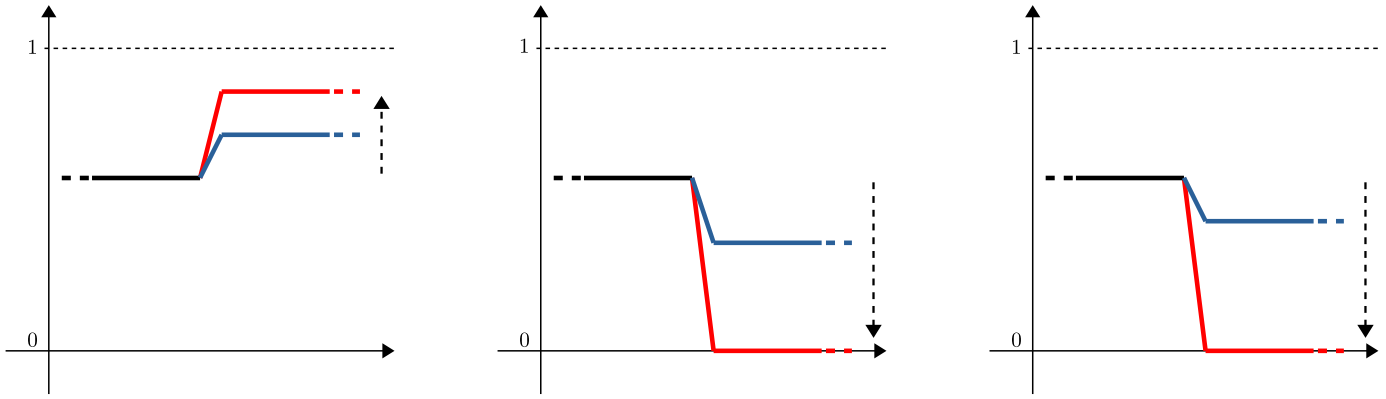
(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.12 Processing time increase effects on Blocking time KPI



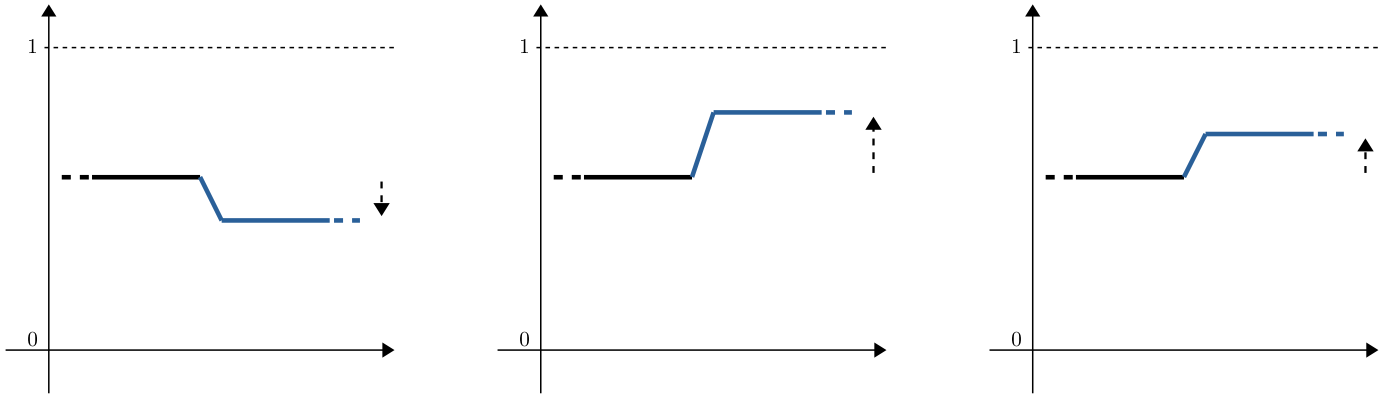
(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.13 Processing time decrease effects on Blocking time KPI



(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.14 Processing time increase effects on Blocking probability KPI



(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.15 Processing time decrease effects on Blocking probability KPI

Effects on Starving_time and Starving_prob KPIs

Starving_time and Starving_prob KPIs behave as follows:

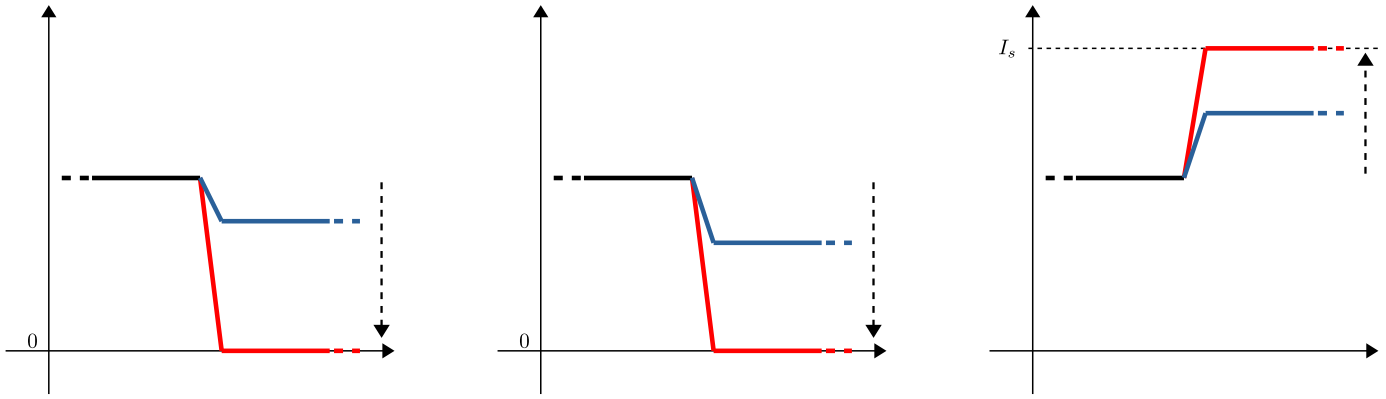
- When a processing time increase occurs ($\mu''_{s_{CH}} > \mu'_{s_{CH}}$)
  - If the system remains stable ($\mu''_{s_{CH}} < \mu_a$)
    - In stages upstream and in correspondence with the changing stage, Starving_time and Starving_prob decrease.
    - In stages downstream and in correspondence with the changing stage, Starving_time and Starving_prob increase.
  - If the system becomes unstable ($\mu''_{s_{CH}} > \mu_a$)
    - In stages upstream and in correspondence with the changing stage, Starving_time and Starving_prob decrease and settle around 0
    - In stages downstream the changing stage
      - Starving_time increases and settles around I_s [3].
      - Starving_prob increases
- When a processing time decrease occurs ($\mu''_{s_{CH}} < \mu'_{s_{CH}}$)
  - In stages upstream and in correspondence with the changing stage, Starving_time and Starving_prob increase.
  - In stages downstream the changing stage, Starving_time and Starving_prob decrease.

The variation extent of Starving_time and Starving_prob in a certain stage s is influenced by stage s position, the processing time variation width ($|\mu''_{s_{CH}} - \mu'_{s_{CH}}|$), and the buffer capacity limits cl_s. It is possible to say that the variation extent is

- wider the larger the processing time variation ($|\mu''_{s_{CH}} - \mu'_{s_{CH}}|$) is
- wider the nearer stage s is with respect to the changing stage s_{CH}
- wider in stages near the changing stage s_{CH}, the higher the buffer capacity limits cl_s are
- narrower in stages far from the changing stage s_{CH}, the higher the buffer capacity limits cl_s are

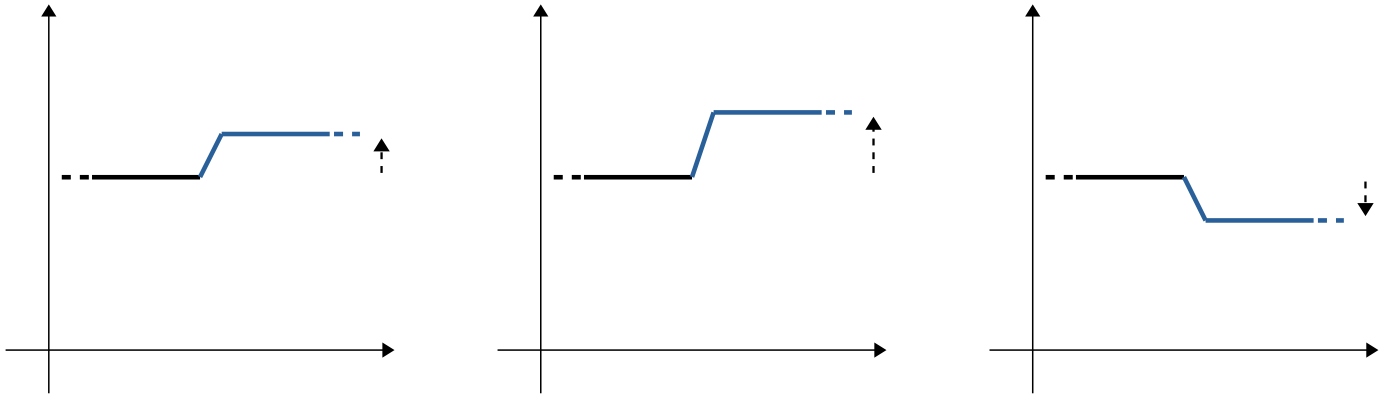
[3] I_s is equal to the difference between average cycle time (indicated by Mid_diff) and average processing time (indicated by Processing_time) of stage s

Starving_time and Starving_prob KPIs usefulness Starving_time and Starving_prob allow to understand how deeply stages different from the changing stage are affected by a processing time variation: Starving_time shows how much time an idle resource waits before a job seizes it, because the previous buffer is empty, while Starving_prob shows how much of its available time a resource on average spends without working because there are no jobs to process.
These KPIs allow to understand a processing time variation direction (increase or decrease) and its approximate size: Starving_time and Starving_prob do not directly show the exact change extent, but high KPI changes are the clue that a big system variation is occurring in a stage.



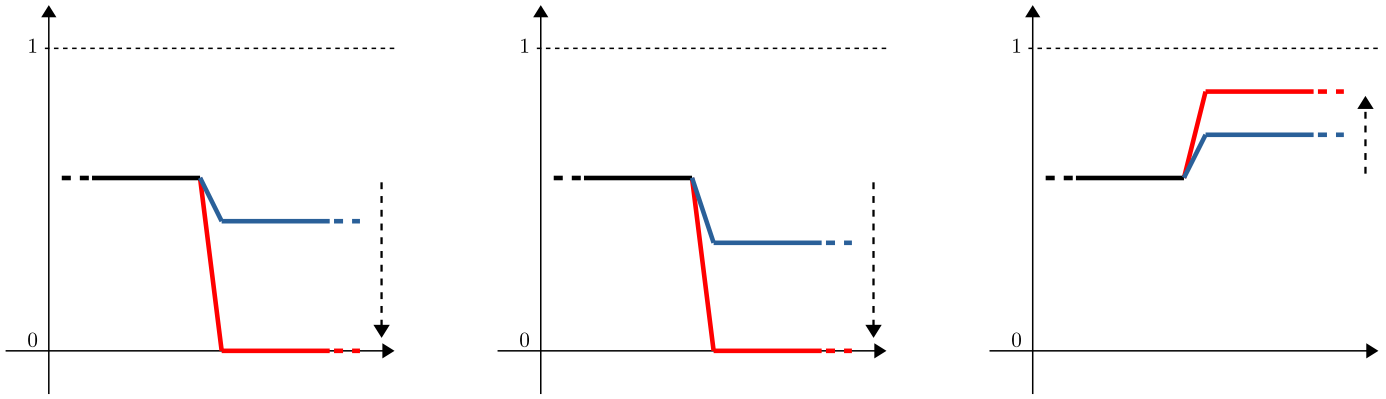
(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.16 Processing time increase effects on Starving time KPI



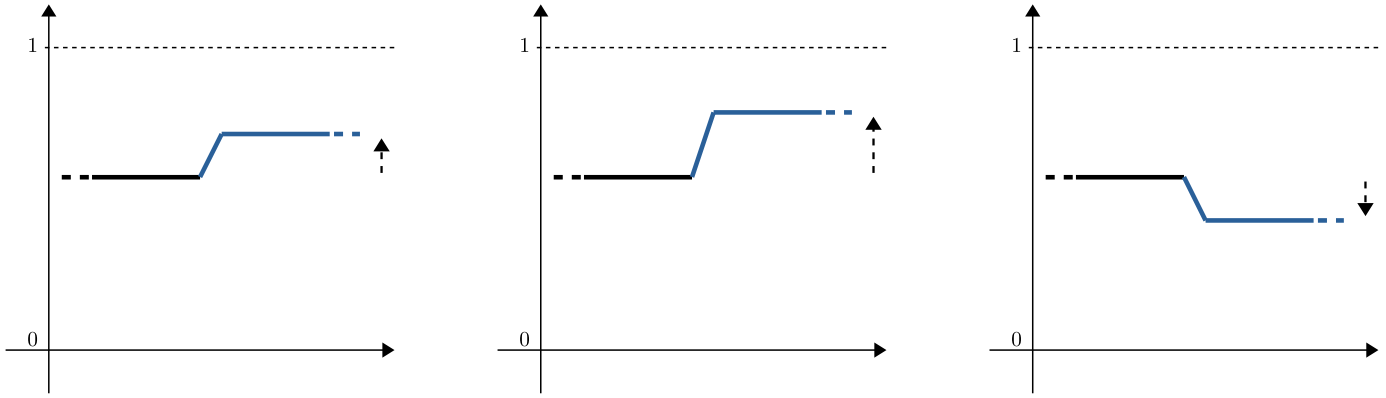
(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.17 Processing time decrease effects on Starving time KPI



(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.18 Processing time increase effects on Starving probability KPI



(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.19 Processing time decrease effects on Starving probability KPI

7.2.2 Buffer capacity variation

In this subsection only Waiting_time, Blocking_time, Starving_time and the respective Stage State KPIs are included. Indeed, as table 7.2 shows, these are the only KPIs influenced by this kind of variation.

Effects on Waiting_time and Average_Queue KPIs

Waiting_time and Average_Queue KPIs behave as follows:

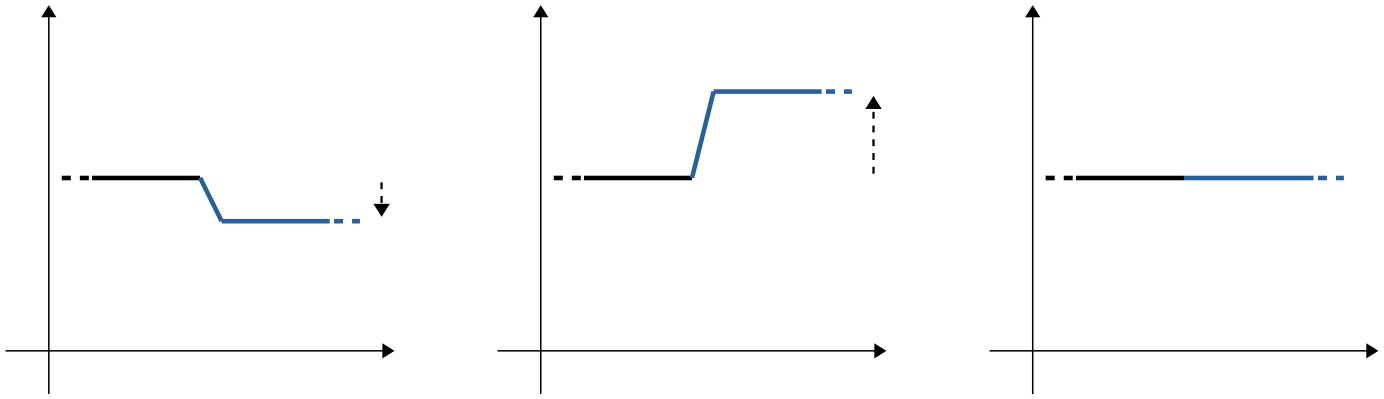
- When a buffer capacity increase occurs ($cl''_s > cl'_s$)
  - In stages upstream the changing stage, Waiting_time and Average_Queue decrease.
  - In the changing stage, Waiting_time and Average_Queue increase.
  - In stages downstream the changing stage, Waiting_time and Average_Queue do not significantly change.
- When a processing time decrease occurs ($cl''_s < cl'_s$)
  - In stages upstream the changing stage, Waiting_time and Average_Queue increase.
  - In the changing stage, Waiting_time and Average_Queue decrease.
  - In stages downstream the changing stage, Waiting_time and Average_Queue do not significantly change.

The variation extent of Waiting_time and Average_Queue in a certain stage s is influenced by stage s position, and the buffer capacity variation width $|cl''_{s_{CH}} - cl'_{s_{CH}}|$. It is possible to say that the variation extent is

- wider the larger the buffer capacity variation ($|\mu''_{s_{CH}} - \mu'_{s_{CH}}|$) is.
- wider the nearer stage s is with respect to the changing stage s_{CH}.
- wider if the changing stage is upstream or in correspondence with the bottleneck ($s_{CH} \leqslant s_{BN}$).
- wider the lower the before-change buffer capacity limits (cl'_s) are.

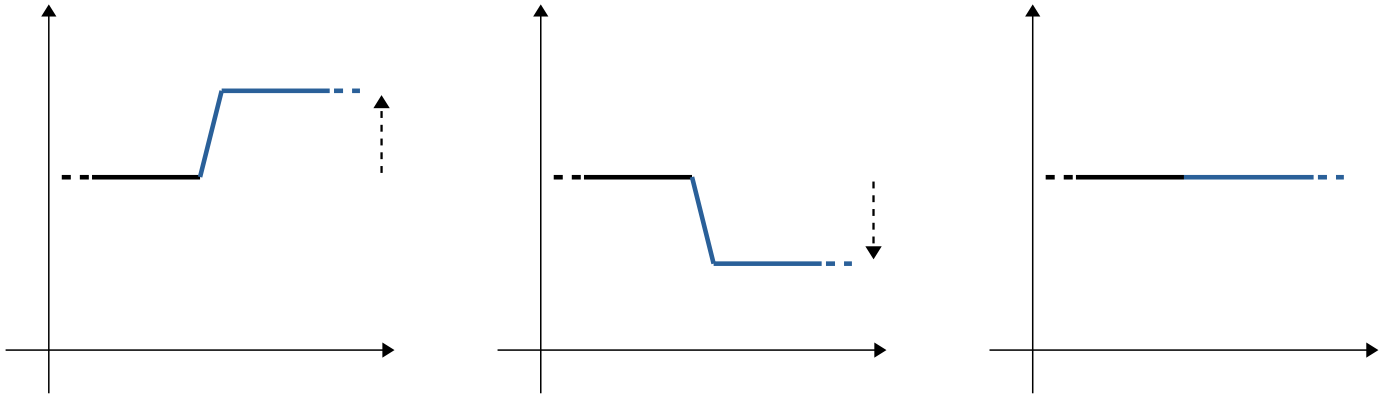
The last observation is linked with what is shown in table 7.2: if cl'_s is too high, and therefore the buffers are underused and mostly unsaturated, a buffer capacity limit variation does not impact Waiting_time and Average_Queue, and so it cannot be detected by these KPIs.

Waiting_time and Average_Queue KPIs usefulness Waiting_time and Average_Queue offer two different perspectives regarding the buffers: Waiting_time has a job-oriented point of view, showing how much time jobs spent in buffers, while Average_Queue is more queue-oriented, allowing to know how many jobs are on average simultaneously present in a buffer.
These KPIs allow to understand a buffer capacity variation direction (increase or decrease) and its approximate size: Waiting_time and Average_Queue do not show the exact change extent, but high KPI changes are the clue that a big system variation is occurring in a stage.



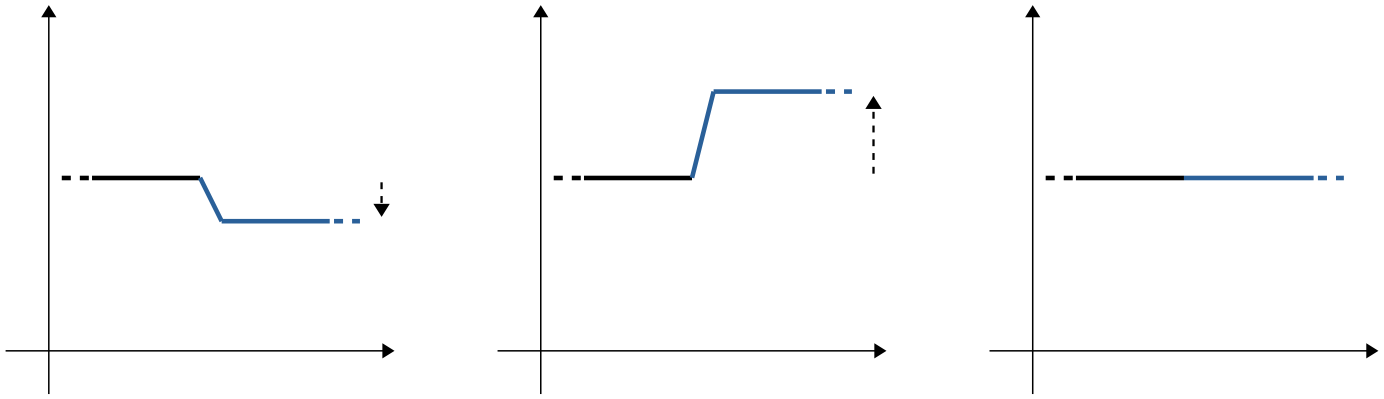
(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.20 Buffer capacity increase effects on Waiting time KPI



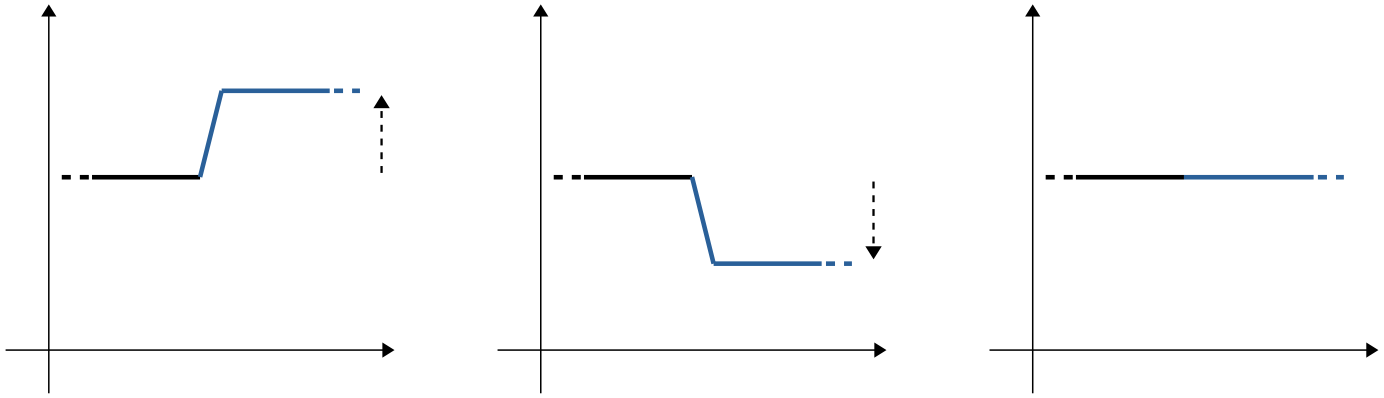
(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.21 Buffer capacity decrease effects on Waiting time KPI



(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.22 Buffer capacity increase effects on Average Queue KPI



(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.23 Buffer capacity decrease effects on Average Queue KPI

Effects on Blocking_time and Blocking_prob KPIs

Blocking_time and Blocking_prob KPIs behave as follows:

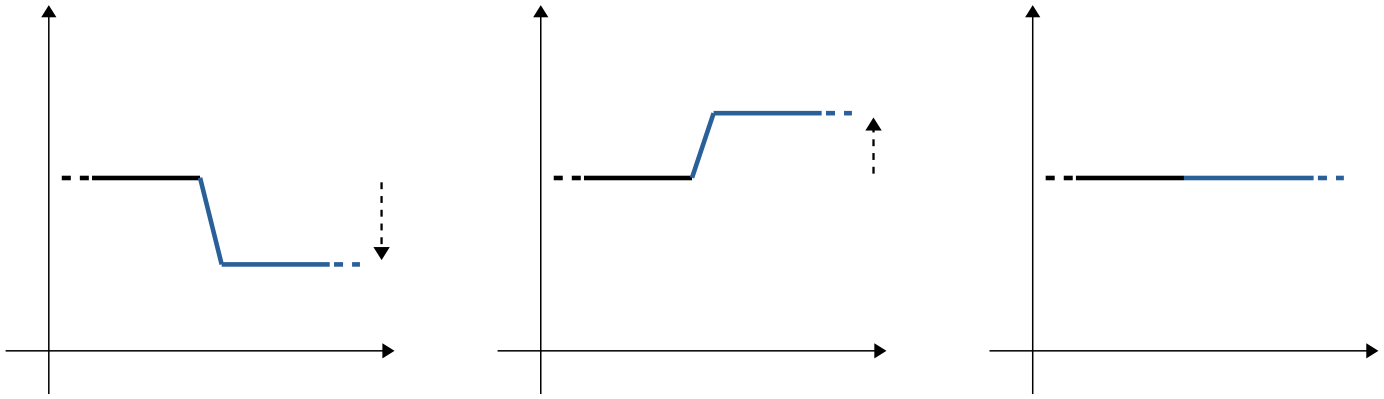
- When a buffer capacity increase occurs ($cl''_s > cl'_s$)
  - In stages upstream the changing stage, Blocking_time and Blocking_prob decrease.
  - In the changing stage, Blocking_time and Blocking_prob increase.
  - In stages downstream the changing stage, Blocking_time and Blocking_prob do not significantly change.
- When a processing time decrease occurs ($cl''_s < cl'_s$)
  - In stages upstream the changing stage, Blocking_time and Blocking_prob increase.
  - In the changing stage, Blocking_time and Blocking_prob decrease.
  - In stages downstream the changing stage, Blocking_time and Blocking_prob do not significantly change.

The variation extent of Blocking_time and Blocking_prob in a certain stage s is influenced by stage s position, and the buffer capacity variation width $|cl''_{s_{CH}} - cl'_{s_{CH}}|$. It is possible to say that the variation extent is

- wider the larger the buffer capacity variation ($|\mu''_{s_{CH}} - \mu'_{s_{CH}}|$) is.
- wider the nearer stage s is with respect to the changing stage s_{CH}.
- wider if the changing stage is upstream or in correspondence with the bottleneck ($s_{CH} \leqslant s_{BN}$).
- wider the lower the before-change buffer capacity limits (cl'_s) are.

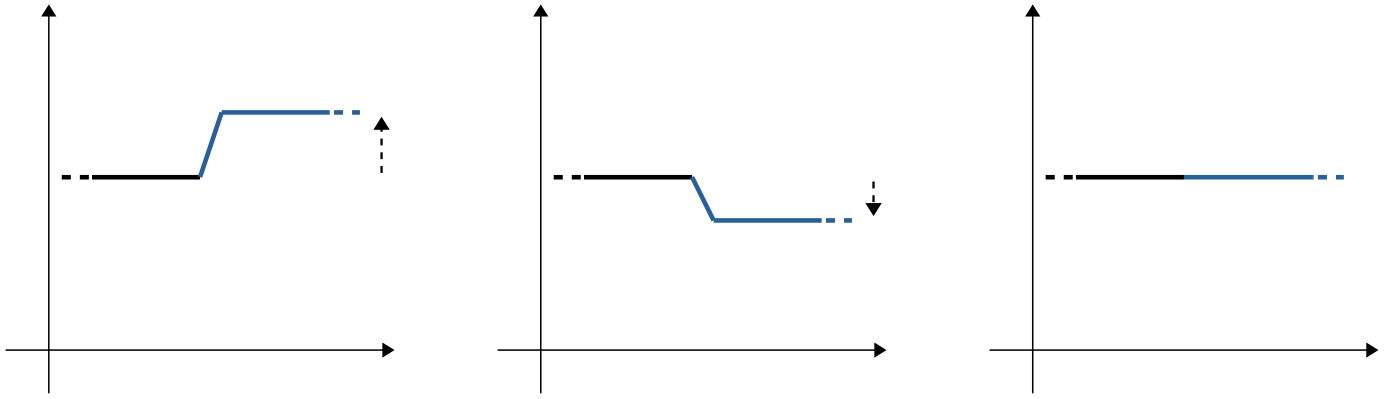
The last observation is linked with what is shown in table 7.2: if cl'_s is too high, and therefore the buffers are underused and mostly unsaturated, a buffer capacity limit variation does not impact Blocking_time and Blocking_prob, and so it cannot be detected by these KPIs.

Usefulness Starving_time and Starving_prob allow to understand how deeply stages different from the changing stage are affected by a buffer capacity variation: Starving_time shows how much time an idle resource waits before a job seizes it, because the previous buffer is empty, while Starving_prob shows how much of its available time a resource on average spends without working because there are no jobs to process.
These KPIs allow to understand a buffer capacity variation direction (increase or decrease) and its approximate size: Starving_time and Starving_prob do not directly show the exact change extent, but high KPI changes are the clue that a big system variation is occurring in a stage.



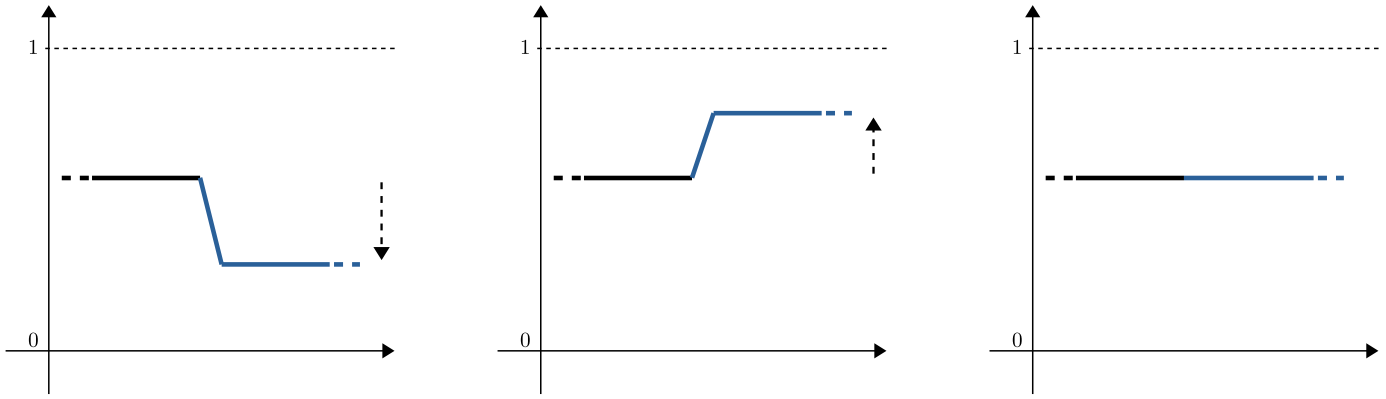
(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.24 Buffer capacity increase effects on Blocking time KPI



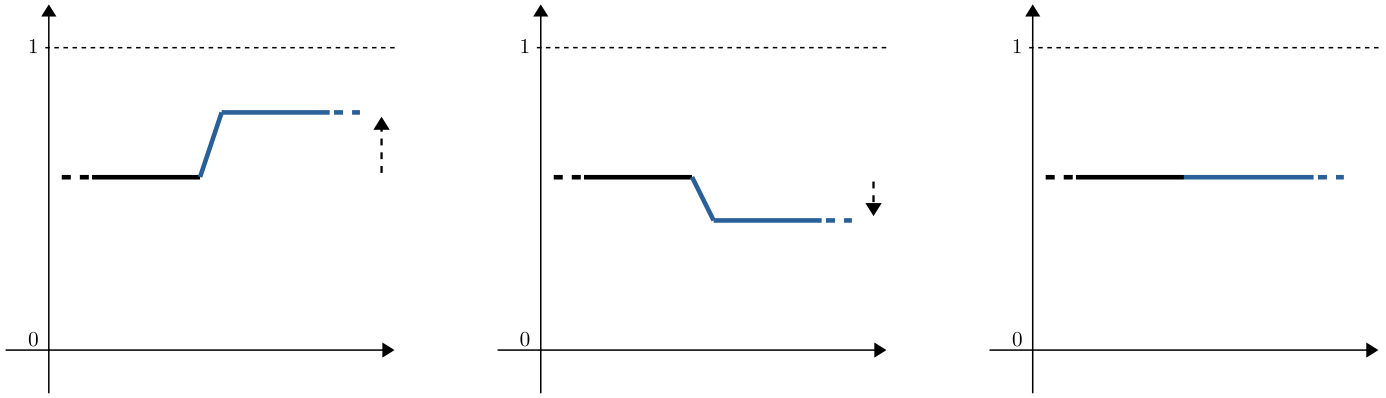
(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.25 Buffer capacity decrease effects on Blocking time KPI



(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.26 Buffer capacity increase effects on Blocking prob KPI



(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.27 Buffer capacity decrease effects on Blocking prob KPI

Effects on Starving_time and Starving_prob KPIs

Starving_time and Starving_prob KPIs behave as follows:

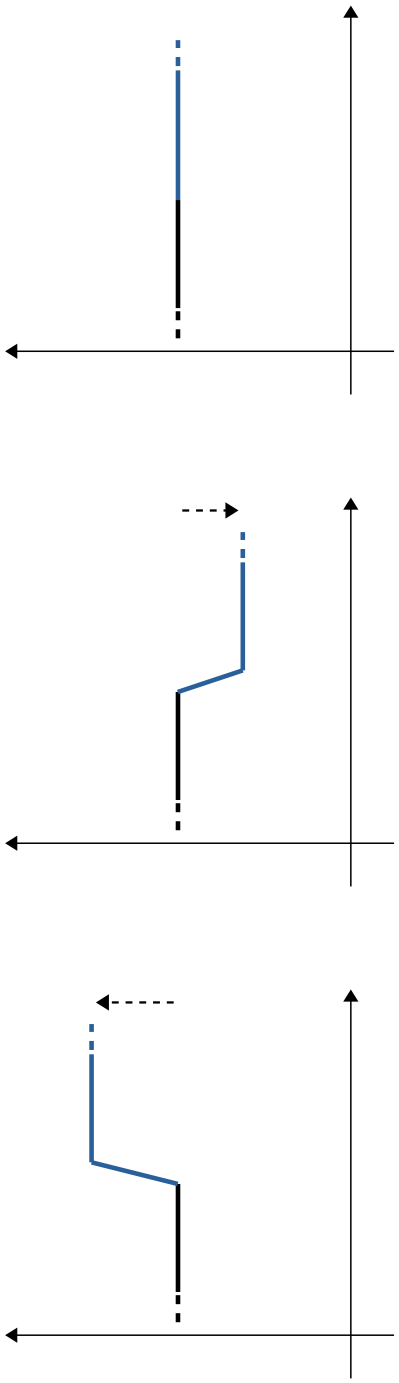
- When a buffer capacity increase occurs ($cl''_s > cl'_s$)
  - In stages upstream the changing stage, Starving_time and Starving_prob increase.
  - In the changing stage, Starving_time and Starving_prob decrease.
  - In stages downstream the changing stage, Starving_time and Starving_prob do not significantly change.
- When a processing time decrease occurs ($cl''_s < cl'_s$)
  - In stages upstream the changing stage, Starving_time and Starving_prob decrease.
  - In the changing stage, Starving_time and Starving_prob increase.
  - In stages downstream the changing stage, Starving_time and Starving_prob do not significantly change.

The variation extent of Starving_time and Starving_prob in a certain stage s is influenced by stage s position, and the buffer capacity variation width $|cl''_{s_{CH}} - cl'_{s_{CH}}|$. It is possible to say that the variation extent is

- wider the larger the buffer capacity variation ($|\mu''_{s_{CH}} - \mu'_{s_{CH}}|$) is.
- wider the nearer stage s is with respect to the changing stage s_{CH}.
- wider if the changing stage is upstream or in correspondence with the bottleneck ($s_{CH} \leqslant s_{BN}$).
- wider the lower the before-change buffer capacity limits (cl'_s) are.

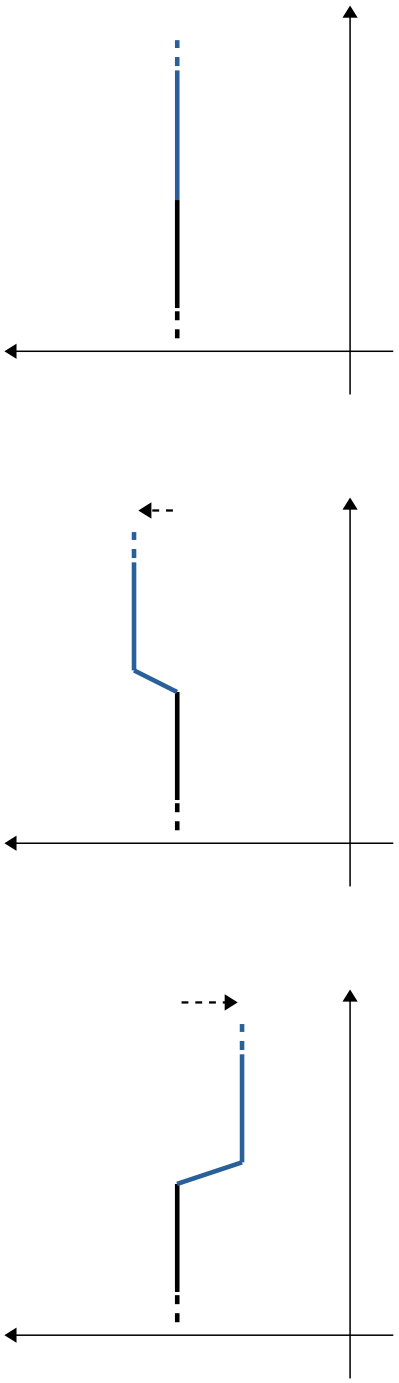
The last observation is linked with what is shown in table 7.2: if cl'_s is too high, and therefore the buffers are underused and mostly unsaturated, a buffer capacity limit variation does not impact Starving_time and Starving_prob, and so it cannot be detected by these KPIs.

Usefulness Starving_time and Starving_prob offer two different perspectives regarding the buffers: Starving_time has a job-oriented point of view, showing how much time jobs spent in buffers, while Average_Queue is more queue-oriented, allowing to know how many jobs are on average simultaneously present in a buffer.
These KPIs allow to understand a buffer capacity variation direction (increase or decrease) and its approximate size: Waiting_time and Average_Queue do not show the exact change extent, but high KPI changes are the clue that a big system variation is occurring in a stage.



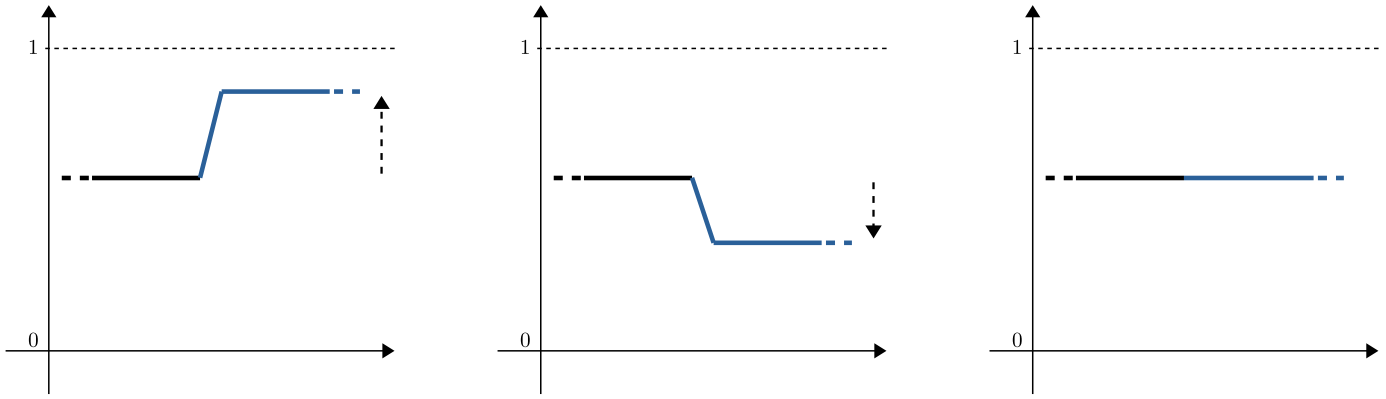
(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.28 Buffer capacity increase effects on Starving time KPI



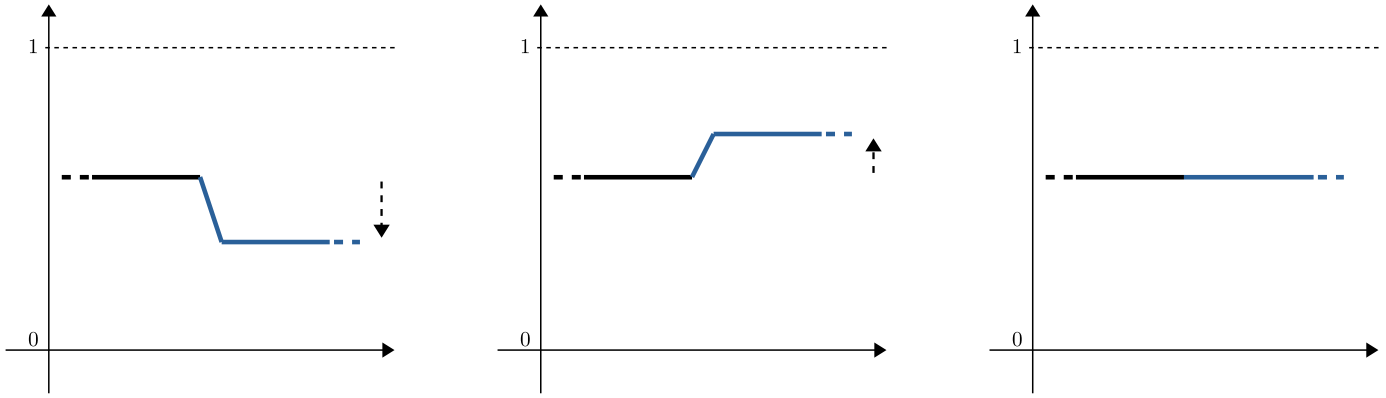
(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.29 Buffer capacity decrease effects on Starving time KPI



(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.30 Buffer capacity increase effects on Starving prob KPI



(a) Stages upstream s_{CH} (b) In correspondence with s_{CH} (c) Stages downstream s_{CH}

Fig. 7.31 Buffer capacity decrease effects on Starving prob KPI

7.3 KPIs from different sensor configurations

Table 7.3 Timestamps necessary to compute each KPI

KPI	Timestamps		
	Timestamp_Buffer T_B	Timestamp_Resource T_R	Timestamp_End T_E
Input_diff	✓		
Mid_diff		✓	
Output_diff			✓
Waiting_time	✓	✓	
Processing_time		✓	✓
Blocking_time	✓		✓
Starving_time	✓	✓	
Average_Queue	✓	✓	
Utilization		✓	✓
Blocking_prob	✓		✓
Starving_prob	✓	✓	

Chapter 8

Conclusions

References

[1] Alok K. Choudhary, Jennifer A. Harding, and Manoj K. Tiwari. Data mining in manufacturing: a review based on the kind of knowledge. 20(5), 2009.

[2] Ray Zhong, Xun Xu, Eberhard Klotz, and Stephen Newman. Intelligent manufacturing in the context of industry 4.0: A review. *Engineering*, 3:616–630, 10 2017.

[3] Martin Wollschlaeger, Thilo Sauter, and Juergen Jasperneite. The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0. *IEEE industrial electronics magazine*, 11(1):17–27, 2017.

[4] Jay Lee, Behrad Bagheri, and Hung-An Kao. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing letters*, 3:18–23, 2015.

[5] Elisa Negri, Luca Fumagalli, and Marco Macchi. A review of the roles of digital twin in cps-based production systems. *Procedia manufacturing*, 11:939–948, 2017.

[6] Xindong Wu, Xingquan Zhu, Gong-Qing Wu, and Wei Ding. Data mining with big data. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):97–107, 2014.

[7] Wil Van Der Aalst, Arya Adriansyah, Ana Karla Alves De Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose, Peter Van Den Brand, Ronald Brandtjen, Joos Buijs, Andrea Burattin, Josep Carmona, Malu Castellanos, Jan Claes, Jonathan Cook, Nicola Costantini, Francisco Curbera, Ernesto Damiani, Massimiliano De Leoni, and Pavlos Delias. Process mining manifesto. volume 99. Springer, 2012.

[8] Wil M. P. van der Aalst. *Process Mining: Data Science in Action*. Springer, Heidelberg, 2 edition, 2016.

[9] Eric Rojas, Jorge Munoz-Gama, Marcos Sepúlveda, and Daniel Capurro. Process mining in healthcare: A literature review. *Journal of biomedical informatics*, 61(C):224–236, 2016.

[10] G. Lugaresi, M. Zanotti, D. Tarasconi, and A. Matta. Manufacturing systems mining: Generation of real-time discrete event simulation models. volume 2019-, pages 415–420. Institute of Electrical and Electronics Engineers Inc., 2019.

[11] Mahendrawathi Er, Noval Arsad, H.M. Astuti, Renny Pradina, and Rivia Utami. Analysis of production planning in a global manufacturing company with process mining. *Journal of Enterprise Information Management*, 31:00–00, 01 2018.

[12] Minjeong Park, Minseok Song, Tae Baek, Sookyoung Son, Seung Ha, and Sung Cho. Workload and delay analysis in manufacturing process using process mining. pages 138–151, 06 2015.

[13] A. Burattin, A. Sperduti, and W. M. P. van der Aalst. Control-flow discovery from event streams. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 2420–2427, July 2014.

[14] M. Hassani, S. Siccha, F. Richter, and T. Seidl. Efficient process discovery from event streams using sequential pattern mining. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 1366–1373, Dec 2015.

[15] Peter Denno, Charles Dickerson, and Jennifer Harding. Dynamic production system identification for smart manufacturing systems. *Journal of Manufacturing Systems*, 48, 05 2018.

[16] R. P. Jagadeesh Chandra Bose, Ronny S Mans, and Wil M. P van Der Aalst. Wanna improve process mining results? In *2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 127–134. IEEE, 2013.

[17] Rodrigo Romero-Silva, Erika Marsillac, Sabry Shaaban, and Margarita Hurtado-Hernández. Serial production line performance under random variation: Dealing with the 'law of variability'. *Journal of manufacturing systems*, 50:278–289, 2019.

[18] A. Schütze, N. Helwig, and T. Schneider. Sensors 4.0 – smart sensors and measurement technology enable industry 4.0. *Journal of sensors and sensor systems*, 7(1):359–371, 2018.

[19] R.P.J.C. Bose, W.M.P. Van Der Aalst, I. Žliobaite, and M. Pechenizkiy. Handling concept drift in process mining. volume 6741, pages 391–405, 2011.

[20] Iñaki Ucar, Bart Smeets, and Arturo Azcorra. simmer: Discrete-event simulation for R. *Journal of Statistical Software*, 90(2):1–30, 2019.

[21] Hadley Wickham, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, Alex Hayes, Lionel Henry, Jim Hester, Max Kuhn, Thomas Lin Pedersen, Evan Miller, Stephan Milton Bache, Kirill Müller, Jeroen Ooms, David Robinson, Dana Paige Seidel, Vitalie Spinu, Kohske Takahashi, Davis Vaughan, Claus Wilke, Kara Woo, and Hiroaki Yutani. Welcome to the tidyverse. *Journal of Open Source Software*, 4(43):1686, 2019.

[22] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020.