DISSERTATION

Fault-Tolerant Clock Synchronization for Embedded Distributed Multi-Cluster Systems

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften
unter der Leitung von

O.Univ.-Prof. Dr.phil. Hermann Kopetz

Institut für Technische Informatik 182

eingereicht an der Technischen Universität Wien,
Fakultät für Technische Naturwissenschaften und Informatik

von

Michael Paulitsch

Matr.-Nr. 9326621

St. Martiner Weg 14, 9210 Pörtschach, Austria

Wien, im September 2002 .....

Fault-Tolerant Clock Synchronization for Embedded Distributed Multi-Cluster Systems

Embedded computer control systems become more and more common for different functional tasks in cars, airplanes, and factory automation systems. In addition, systems for different functional tasks become interconnected and system functions more integrated. As a consequence, these systems become larger and more complex. Multi-cluster systems are a way to structure large distributed systems to manage complexity and overcome communication bandwidth limitations. A cluster is a system where nodes are tightly coupled and are connected via direct communication links. Nodes of different clusters are loosely coupled and are connected via special nodes that connect two clusters, so-called gateway nodes.

In distributed control systems, a common notion of time of all nodes is fundamental for control, because it allows a consistent system view of dynamic environments and supports meaningful exchange of time-related data between clusters. This thesis presents a clock synchronization algorithm for multi-cluster systems that is especially suited for embedded control systems. The requirements of embedded systems are addressed by supporting increased dependability necessities and decreased computing resources of embedded systems compared to desktop computer systems. Furthermore, the algorithm assures the composability of the cluster time bases, that is the precision of clusters is not worsened when several clusters are connected and synchronized.

By addressing systematic and stochastic errors of cluster times differently, the influence of systematic errors is eliminated and the quality of synchronization only depends on stochastic errors. Since systematic errors of cluster times are usually an order of magnitude larger than stochastic errors for typical real-time embedded control systems, the presented algorithm achieves a significant improvement to known synchronization algorithms.

An implementation of the proposed clock synchronization algorithm on top of the Time-Triggered Architecture and experiments show that clock synchronization of nodes of multi-cluster systems can be achieved with an accuracy of less than one microsecond.

Fehlertolerante Uhrensynchronisation für eingebettete verteilte Multi-Cluster-Systeme

Eingebettete rechnergestützte Steuerungssysteme werden immer verbreiteter für verschiedene Aufgaben in Autos, Flugzeugen oder Automatisierungssystemen eingesetzt. Der Trend geht dahin, die unterschiedlichen Systeme zu vernetzen und mehr Funktionen zu integrieren. Als Konsequenz werden solche Systeme aufwendiger und komplizierter. Multi-Cluster-Systeme sind ein Ansatz zur Strukturierung von großen verteilten Systemen, um die Komplexität beherrschen und die Bandbreitenlimitierung bewältigen zu können. Ein Cluster ist ein System, dessen Rechenknoten eng gekoppelt und direkt kommunizieren können. Rechenknoten von verschiedenen Clustern sind lose gekoppelt und können nur über sogenannte Gateways miteinander kommunizieren. Gateways sind spezielle Rechenknoten, die zwei Kommunikationssysteme miteinander verbinden.

Ein gemeinsamer Zeitbegriff ist für die koordinierte Steuerung in verteilten Steuerungssystemen notwendig, weil ein gemeinsamer Zeitbegriff eine konsistente Systemsicht erlaubt und den sinnvollen Datenaustausch von zeitabhängigen Variablen ermöglicht. Diese Arbeit präsentiert einen Uhrensynchronisationsalgorithmus für Multi-Cluster-Systeme, der für den Einsatz in eingebetteten Systemen geeignet ist. Die Anforderungen an eingebettete Systeme sind vor allem hohe Zuverlässigkeit und geringer Ressourcen-Gebrauch im Vergleich zu Desktop-Computersystemen. Der vorgestellte Algorithmus erreicht die Integration von Zeitbasen von Clustern, wobei die Präzision von Zeitbasen der Cluster durch die Integration von mehreren Clustern nicht verschlechtert wird.

Systematische Uhrenfehler beeinflussen die Synchronisationsqualität nicht, weil systematische und stochastische Uhrenfehler unterschiedlich behandelt werden. Da systematische Uhrenfehler meist eine Zehnerpotenz größer als stochastische sind, erreicht der Algorithmus eine signifikante Verbesserung in der Synchronisationsqualität verglichen mit bekannten Uhrensynchronisationsalgorithmen.

Eine Implementierung des Uhrensynchronisationsalgorithmus unter Verwendung der Time-Triggered Architecture sowie Experimente zeigen, dass Uhren von Rechenknoten verschiedener Cluster mit einer Präzisionsgenauigkeit, die kleiner als eine Mikrosekunde ist, erreicht werden kann.

Contents

1 Introduction **1**

1.1 Motivation and Objectives 1

1.2 Structure of the Thesis 2

2 Concept and Terms **5**

2.1 Distributed Systems 5

2.1.1 Dependability in Distributed Systems 10

2.1.2 Aspects of Distributed Systems 20

2.1.3 Types of Distributed Systems 22

2.2 Clocks, Time, and Clock Synchronization 26

2.2.1 Concepts of Clocks 26

2.2.2 Clock Synchronization 29

2.2.3 Time Standards and Sources 35

2.2.4 Time Aspects from an Application-Specific View 37

2.3 Time-Triggered Architecture 39

2.4 Summary 42

3 Problem Statement and Objectives **45**

3.1 Problem Statement 45

3.2 Objectives 46

3.3 Summary 48

4 Framework and Assumptions **49**

4.1 Framework 49

4.1.1 Flow of Timing Information 51

4.2 Architecture of One Cluster 53

4.3 Requirements 54

4.3.1 Communication System Requirements 54
4.3.2 Synchronization-Related Requirements 54
4.3.3 Tolerance to Faulty Relational Clock Times 55
4.3.4 Requirements Regarding the Drift Rate and Correction of the Global Time of a Cluster 56
4.4 Summary 57

5 Multi-Cluster Clock Synchronization 59
5.1 Principle of Operation 60
5.2 Analysis of Synchronization 64
5.2.1 Maximum Drift and Correction Rates 65
5.2.2 Compensation of Systematic Part of the Drift Rate . . . 65
5.3 Non-Interference 68
5.3.1 Consistent Agreement on One Correction Value 68
5.3.2 Correction Does not Interfere with Precision 69
5.3.3 Remaining Influence Due to Imperfect Synchronization . 70
5.4 Self-Stabilization 71
5.5 Resource Requirements 73
5.6 Discussion of Algorithm Parameters 74
5.7 Summary 74

6 Multi-Cluster Clock Synchronization in the Time-Triggered Architecture 77
6.1 Goals 78
6.2 Implementation 78
6.3 Analysis of the Accuracy 80
6.4 Analysis of the Influence of the Algorithm on the Precision . . 83
6.5 Analysis of Self-Stabilization 84
6.6 Summary 85

7 Experimental Evaluation 87
7.1 Experimental Setup 88
7.2 Prerequisites 88
7.3 Long-Term Evaluation of Accuracy 91
7.4 Independence of the Accuracy from the Value of the Systematic Part of the Drift Rate 95

7.5 Change of Systematic Drift Rate 97
7.6 Changing History Length 99
7.7 Changing Measurement Parameters 103
7.8 Non-Interference 106
7.9 Summary 109

8 Conclusion **111**

Bibliography **115**

A Notation **131**

B Measurement Data **133**

Curriculum Vitae **143**

List of Figures

2.1 Dependability tree 10
2.2 Failure classification 12
2.3 Fault classification 13
2.4 Fault-error-failure-model 14
2.5 Concept of a virtual clock 28
2.6 Optimal representation of time in real-time systems 37
2.7 Sparse time 38
2.8 Time-Triggered Architecture system 39
2.9 TDMA round (four nodes) 40
2.10 TTP/C cluster (star architecture) with four nodes and two guardians 41

4.1 Multi-cluster system 50
4.2 Propagation of timing info in a multi-cluster system 52
4.3 Cluster with eight nodes, three of the nodes are time master nodes 53

5.1 Pseudo code describing the computation of the correction value 62
5.2 Pseudo code describing the time master operations 64
5.3 Pseudo code describing the integration 65
5.4 Deviation of global time of a cluster from a relational clock time using different correction techniques 66

6.1 Implementation of a time master node with the Time Capturing Unit (TPU) and a GPS Receiver as external time source using a TTPNode 79

7.1 Density distribution of the drift rate of a synchronized virtual clock 90
7.2 Deviation of virtual clock times from relational clock time without calibrating 91

7.3 Deviation of virtual clock times from relational clock time after calibrating 92
7.4 External synchronization: deviation of virtual clock time from relational clock time 94
7.5 Inter-cluster synchronization: deviation of virtual clock time from relational clock time 95
7.6 External synchronization: deviation of virtual clock time from relational clock time 96
7.7 External Synchronization: deviation after a change in the set of nodes used for internal clock synchronization 98
7.8 Inter-cluster synchronization: deviation of virtual cluster time from relational clock time with changing history length 100
7.9 External synchronization: deviation of virtual clock time from relational clock time with changing history length 101
7.10 External synchronization: standard deviations 102
7.11 Inter-cluster synchronization: standard deviation of deviation values 104
7.12 Inter-cluster synchronization: range of deviation values 105
7.13 Frequency of time difference values (external clock synchronization running) 107
7.14 Frequency of time difference values (external clock synchronization not running) 108

List of Tables

7.1 Frequency of values representing the difference between expected and actual arrival time of messages (in percent) (external clock synchronization running) 107

7.2 Frequency of values representing the difference between expected and actual arrival time of messages (in percent) (external clock synchronization not running) 108

A.1 Overview of notation 132

B.5 External Synchronization: deviation after a change in the set of nodes used for internal clock synchronization 135

B.1 Frequency of measured clock drift rate values of global time . . 136

B.2 External synchronization: deviation of virtual clock time from relational clock time 137

B.3 Inter-cluster synchronization: deviation of virtual clock time from relational clock time 138

B.4 Frequency of deviation values 139

B.6 Inter-cluster synchronization: deviation of virtual cluster time from relational clock time with changing history length 140

B.7 External synchronization: deviation of virtual clock time from relational clock time with changing history length 141

B.8 External synchronization: standard deviations 142

B.9 Inter-cluster synchronization: standard deviation of deviation values 142

B.10 Inter-cluster synchronization: range of deviation values 142

Chapter 1

Introduction

"Where shall I begin, please your Majesty?" he asked.
"Begin at the beginning," the King said, gravely,
"and go on till you come to the end: then stop."

Alice's Adventures in Wonderland, Lewis Carroll

Time-triggered system architectures, such as the Time-Triggered Architecture (TTA) [KB02], SPIDER [Min00], SAFEbus [ARI93, HD93], or Flexray [MHB+01], are used in embedded control systems for safety-critical control applications in cars and airplanes. Currently, these embedded control systems are small and all computing nodes are directly connected. As embedded control systems become larger and more complex, computing nodes must be grouped in clusters – so called multi-cluster systems – due to limited communication bandwidth and complexity management. The coupling of nodes within a cluster will be tight, while the coupling between clusters will be loose.

1.1 Motivation and Objectives

A precise time base in distributed control systems is fundamental for control, as manifests in recent efforts of the instrumentation and measurement industry to establish a common standard for synchronization [IEE02]. Consequently, in embedded control systems, nodes of a cluster need a common fault-tolerant time base, that is, nodes of a cluster must perform internal clock synchronization. In order for different clusters to be able to collectively perform computational

activities to control an object or a process, time bases of different clusters must be put into relation to each other, that is, cluster time bases must be synchronized with respect to each other. A common cluster-wide notion of time allows a consistent system view of dynamic environments.

Moreover, when an alpha particle hits a chip[1] with a feature size in the sub-micron range, the whole chip may fail. This is in contrast to the experience of chips with feature sizes in the micron range where only small units of a chip fail due to transient faults, such as alpha particles. Current fault tolerance approaches address transient faults at a chip level, because the effect of a transient fault effects only a small part of the chip and can be handled at locally within the effected area of the chip. With chips in the sub-micron range current fault tolerance strategies do not work anymore and an approach with replication at system level is needed, such as employment of distributed systems with loosely coupled nodes. A system-wide common notion of time enables the introduction of transparent fault tolerance [Bau01] and allows tolerance of transient faults at the cluster or multi-cluster level.

This thesis extends existing clock synchronization concepts for single-cluster systems to multi-cluster systems by developing and evaluating a fault-tolerant clock synchronization algorithm for multi-cluster systems especially suited for embedded control systems. Furthermore, it validates the approach by analysis and measurement. The proposed approach is different from existing approaches as embedded control systems are part of a larger system and are often interconnected with other systems. They are distinct from desktop computer systems as they have high dependability requirements due to greater autonomy needs compared to desktop computers and limited hardware resources. These differences lead to specific requirements a synchronization algorithm must meet.

This thesis addresses problems of the overarching research themes *predictability and manageability* and *inter-operability* of embedded networks as categorized by the U.S. National Research Council [BCE+01].

1.2 Structure of the Thesis

This thesis is organized as follows: In Chapter 2, we will demarcate the area of distributed systems by defining and explaining the relevant terms and concepts of this scientific field with emphasis on multi-cluster, embedded, and real-time systems. We will also introduce different concepts of clocks and describe concepts of clock synchronization and time standards. We will end the chapter

[1] We assume that a node consists of one major chip. As a consequence, an error of a chip implies an error of a node.

with a survey of the Time-Triggered Architecture, which is used as reference implementation platform in this thesis.

Chapter 3 will address the problem this thesis tackles. We will give a precise problem specification and present a requirements analysis of clock synchronization algorithms for embedded control systems. Known clock synchronization algorithms that are related to the presented research work are also discussed in this chapter.

Any concept abstracts from irrelevant facts and bases on assumptions. Chapter 4 will describe the framework of this thesis and the prerequisites of the algorithm.

Chapter 5 will describe the principle of multi-cluster clock synchronization. The presented algorithm is round-based and addresses the systematic and the stochastic error differently. The common notion of time and atomic broadcast will avoid interference between node-external parameters and the precision of the cluster. We will also analyze the maximum drift and correction rates of the synchronization and the non-interference and self-stabilization properties. The chapter will also look into the resource requirements of the algorithm and discuss algorithm parameters.

The presented concepts are validated using the Time-Triggered Architecture as reference platform. Chapter 6 will describe this implementation. It will start with a presentation of the goals of the implementation. We will then analyze the achievable accuracy of the algorithm and the influence of the algorithm on the precision of a cluster. Furthermore, we will look into self-stabilization of the implementation.

In Chapter 7, we will evaluate the algorithm using measurements. We will start with a brief description of the experimental setup and measurements that validate that assumptions of a constant systematic clock error and a symmetrically distributed stochastic one hold. Based on these results, we will present measurements of the accuracy in a long-term evaluation of the implementation. Experiments that examine the elimination of the systematic part of the drift rate and different algorithm-specific parameters will be presented. The chapter will end with measurements concerning the influence of the algorithm on the precision of a cluster.

Finally, this thesis ends with a conclusion in Chapter 8 summarizing the key results of the presented work and presenting an outlook on the future research in this area.

Chapter 2

Concept and Terms

Ohne Zweifel war es einer der genialsten Gedanken des Menschen,
was der Inbegriff des Flüchtigen, was nicht zu sehen
und nicht unmittelbar zu begreifen ist, die Zeit.

Thomas Mann (1875-1955)

In the area of distributed computer systems and clock synchronization, the semantics of used terms differs due to the different range of application and purposes of distributed systems and clock synchronization. Introducing clear concepts and a consistent notion with respect to terminology is inevitable for a discussion about the concepts. This chapter introduces the necessary terms and concepts.

This chapter first focuses on distributed systems and dependability of distributed systems. It, then, presents aspects and types of distributed systems. The second major part introduces the reader to concepts of clocks, time standard and formats, and clock synchronization. The chapter ends with an overview of the Time-Triggered Architecture.

2.1 Distributed Systems

Various definitions of the technical term "distributed system" have been given in the literature. The following two definitions contain important aspects of distributed systems:

For Schroeder, *"a distributed system is several computers doing something together. Thus, a distributed system has three primary characteristics: multiple computers, interconnections, and shared state"*[Sch93b].

Coulouris et alii *"define a distributed system as a collection of autonomous computers linked by a network, with software designed to produce an integrated computing facility"* [CDK94].

Nodes, Communication Systems, Software, and States. These definitions name the following basic elements: A distributed system consists of multiple, autonomous computers, which we call *nodes* in this thesis. A network interconnects these nodes. We call this network *communication system*. The *software* is an algorithmic description that determines the behavior of the system and coordinates activities towards a common goal. For this purpose, the relevant parts of local *state*s are exchanged via messages. These relevant parts of the local states are the *shared state* of the distributed systems. At a given instant, the *state* of a node is the values assigned to an internal data structure of this node that synthesizes all cumulative effects of all received messages and input operations at all input interfaces between the startup of the system and this given instant [JKK+01]. The state enables the determination of a future output solely on the basis of the future input and the state the system is in [MT89]. Software on its own does not have a state [Szy99]. Each node of a distributed system and its software cooperate to maintain a shared state. Put it another way, if the correct operation is described in terms of some global invariants, then maintaining those invariants requires the correct and coordinated operation of nodes and its software [Sch93b]. The *global state* of a system is defined as the union of the local states of its components [Sch93a].

Components. We use the term *component* for referring to a constituent part of a distributed system, that is to a node, the software running at this node, and the node's state. At a given level of abstraction, the term 'system' is equivalent to the term 'component' with only one distinction: while a system can be decomposed into subsystems, a component cannot be reasonably decomposed for a given purpose of abstraction.

Characteristics for Employment. The following six characteristics are the key reasons for the employment of distributed systems [CDK94]:

Resource sharing. Resource is an abstract term that characterizes hardware components such as printers, disks, and I/O devices, and software-defined entities, such as files and databases. Resources are physically located at

nodes. A software module, called the *resource manager*, enforces the separate management policies and methods for resources or groups of resources. It also provides common requirements for the shared exploitation of resources, such as naming, access, and consistency management. Depending on the interaction style between resource managers and resource users, one can distinguish two models: the *client-server model* [AP95] and the *object-based model* [Weg84, Mye88]. In the client-server model, all shared resources are held and managed by servers. Clients issue requests to servers whenever they need to access one of their resources. In the object-based model, each shared resource is viewed as an object. An object has a fixed identity though it can move through the network. An access of the resource occurs by sending a message containing the request to the corresponding object.

Openness. The openness of a distributed system addresses the possible extensibility of this system. That is the degree to which shared resources and nodes can be added without disruption of service to the existing system. A system can be open with respect to hardware (such as the augmentation of a new node) or software (the introduction of a new service). Through the specification and documentation of key interfaces, the openness of a system can be achieved. To achieve this, each node and resource must adhere to the published standard, which is also called the *architectural style* of a system [JKK+01].

Concurrency. The execution of a program in an environment together with at least one thread of control is called a *process*. If several processes exist at the same time in a system, we say that they are executed concurrently. The reasons of concurrency are for example the exploitation of logical parallelism or the concurrent use of resources by different users. With the introduction of concurrency in systems, mechanisms for controlling concurrency must be established in order to avoid problems associated with concurrent execution. The design issues to be encompassed for concurrency include the communication among processes, sharing of and competing for resources, synchronization of activities of multiple processes, and allocation of processor time to processes [Sta01]. There are several approaches to synchronizing concurrent accesses to shared resources, such as semaphores [Dij68], non-blocking write [KR93], monitors [Ben90], and message passing [Sta01].

Scalability. *"A* scalable *distributed system is one that can easily cope with addition of users and sites, and whose growth involves minimal expense, performance degradation, and administrative complexity"* [Sat93]. The need for scalability is not just a problem of node, software, or commu-

nication system performance. According to Satyanarayanan [Sat88], the effects of scale on a distributed system are manifold: considerations of performance and operability dominate the design; security becomes a serious concern; functionally specialized mechanisms rather than general-purpose solutions become more attractive; and the system is likely to be composed of diverse elements, rather than a single homogenous set of elements. The demand for scalability in distributed systems has lead to a design methodology in which no single resource is assumed to be in restricted supply. As demand for a resource grows, it should be possible to extend the system to meet it. Yet, scalability does not cover an arbitrary size of a system; the intended range of scale of the system will be a fundamental influence on its design [Sat88]. For a scalable system, its complexity plays a major role for its design. *"The* complexity *of a system relates to the number of parts, and the number and types of interactions among the parts, that must be considered to understand a particular function of a system. The effort required to understand* any particular function *should remain constant, and* independent *of the system size"* [Kop97] in order to understand all functions of a system and to enable a scalable system design.

Fault tolerance. Every computer system will eventually fail. Fault Tolerance are the methods and techniques that aim at providing the intended system behavior in spite of faults. In distributed systems, it is often appropriate to define *fault containment regions*. That are sets of components that are considered to fail as atomic units, and in a statistically independent way with respect to other fault containment regions [JKK+01]. If a distributed system can be designed in a way that a node is a fault containment region, nodes can be replicated to provide fault-tolerant system behavior. This design approach is called *hardware redundancy*. The allocation of redundant nodes that is required for fault tolerance can be designed so that the hardware is exploited for non-critical activities when no faults are present. This allocation and the replication of nodes, however, adds complexity and costs even if no faults occur.

The second basic design approach to achieve fault tolerance is *software recovery*. There are two types of recovery: *forward* and *backward* recovery [Jal94]. For backward recovery, the state of a node or defined unit is periodically saved to permanent storage. This is called a checkpoint. If an error is detected, the system state is recovered from an earlier state (or – in other words – "rolled back" to the last checkpoint). For forward recovery, if an error in the state is detected, the system attempts to "go forward" and try to make the state error-free by taking the necessary corrective actions. This requires an accurate assessment to be made of the

damage to the state, assumptions are required about the nature of the damage or error, and often redundant information in the state for the detection of an error. Section 2.1.1 will cover fault tolerance in distributed systems in more detail.

Transparency. Transparency is the concealment of components in a distributed system, so that the system is perceived as a whole rather than as a collection of independent components. The ANSA Reference Manual [ANS89] and the International Standards Organization's Reference Model for Open Distributed Processing [Int92, dM95] identify eight forms of transparency. These reflect the different motivations and goals of distributed systems. The eight forms of transparency are:

- *Access transparency* enables local and remote information objects to be accessed using identical operations.
- *Location transparency* enables information objects to be accessed without knowledge of their location.
- *Concurrency transparency* enables several processes to operate concurrently using shared information objects without any unwanted effects between them.
- *Replication transparency* enables multiple instances of information objects to be used to increase reliability and performance without knowledge of the replicas by users or application programs.
- *Failure transparency* enables the concealment of faults, allowing users and application programs to complete their tasks despite of failure of hardware or software components.
- *Migration transparency* allows the movement of information objects within a system without affecting the operation of users or application programs.
- *Performance transparency* allows the system to be reconfigured to improve performance as loads vary.
- *Scaling transparency* allows the system and applications to expand in scale with change to the system structure or the application algorithms.

In literature, there is often a confusion about the terms *abstraction* and *transparency*. Abstraction *allows not deal with details* while transparency *does not deal with details*. There is evidence [WWWK97] that true transparency is not always a desired characteristic in distributed systems, but should be replaced by respective abstraction [Bau01]. In [Kop00b], Kopetz addresses this point when he categorizes component interfaces

of distributed real-time systems (see Section 2.1.3) from an application-specific view. The three interfaces types are the *real-time-service* interface, the *diagnostic and management* interface, and the *configuration and planning* interface. An interface of the first type provides timely real-time services to the component environment during the operation of the system. The second and third type of interfaces are used for the purpose of internal fault diagnosis and configuration of the distributed system. For real-time service interfaces, transparency is demanded. For the other two interfaces the abstraction and *not* the transparency of a distributed system architecture is desired.

Distributed Versus Parallel Systems. According to Fischer [Fis90], the uncertainty introduced by unreliable communication and the fact that some nodes may be faulty is the main characteristics that enables the class of "distributed" systems to be distinguished from that of "parallel" systems [Pow94].

2.1.1 Dependability in Distributed Systems

According to Carter [Car82], *"computer system dependability may be defined as the trustworthiness and continuity of computer system service such that reliance can justifiably be placed upon this service"*. The service delivered by a system is its behavior as it is perceived by its user(s); a user is another system (human or physical) which interacts with the former [Lap92].

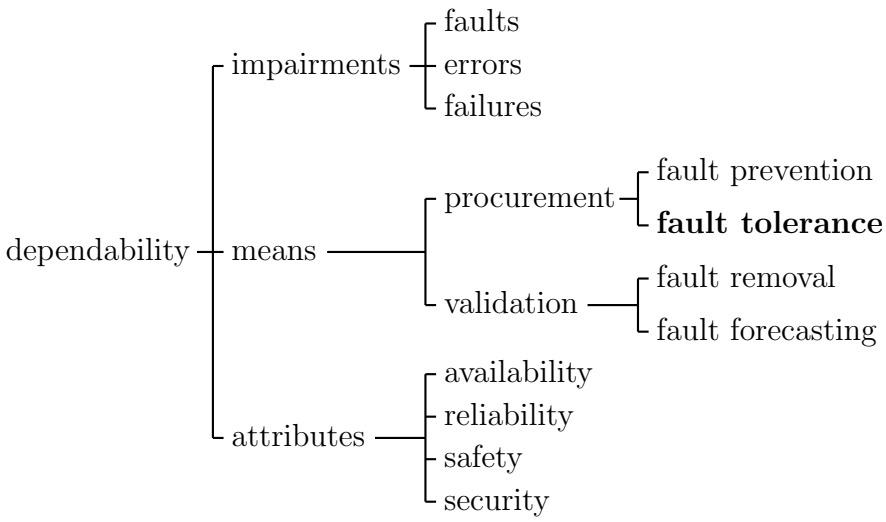


Figure 2.1: Dependability tree [Lap95].

Figure 2.1 depicts the different viewpoints from which one can see dependability. These viewpoints can be grouped into three classes [Lap95, Lap92]:

Impairments. The impairments to dependability are faults, errors, and failures. Impairments are undesired circumstances that lead to or result from a service with missing reliance. Impairments are expected to occur or to be part of any system, since no system can be designed or operated perfectly.

Means. The means for dependability are methods and techniques enabling one (a) to provide the ability to deliver a service on which reliance can be placed, and (b) to reach confidence in this ability.

Attributes. The attributes of dependability (a) enable the properties that are expected from the system to be expressed, and (b) allow the system quality resulting from the impairments and means opposing to them to be assessed.

The impairments to, the means for, and the attributes of dependability are described in more detail in the next paragraphs, because a clear notion of these concepts is necessary to correctly understand the concepts developed in this thesis.

Impairments to Dependability

Failures, errors, and faults impair the dependability of systems.

Failure. A *failure* of a system is an event that denotes a deviation between the actual service and the specified or intended service that is required by its specification [Kop97, LA90]. That is, a system fails when it cannot provide the specified service. If the service of a system meets its specification but deviates from the user's intended behavior, we speak of a specification fault. Systems fail in different ways. These different ways of failing are called their *failure modes* [Lap92]. The failures are classified according to their domain (also called nature), their perception, their consequences (or effects), and their oftenness. Figure 2.2 depicts the different failure modes.

With respect to *failure domain*, value and timing failures are distinguished. A *value failure* means that the value of the delivered service does not comply with its specification. If the timing of the service delivery does not comply with its specification, this is a *timing failure*. Failures in the time domain are also denoted as *crash* failures, if the delivery of service is delayed infinitely. A

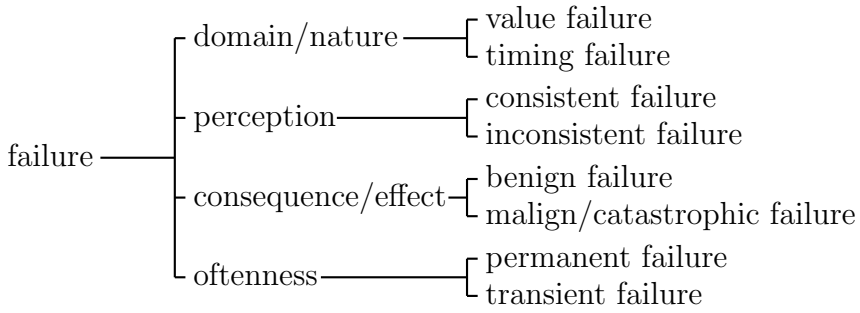


Figure 2.2: Failure classification [Lap95, Kop97].

components exhibits an *omission* failure, if the service delivery to a request is omitted [Cri91].

The *perception* of a failure can be different for different components in a distributed system. If different components of a distributed system perceive different (incorrect) results, this is called an *inconsistent failure*. If all components perceive the same incorrect result, this is called a *consistent failure*. An inconsistent failure is also called *two-faced failure*, *malicious failure*, *asymmetric failure*, or *Byzantine failure* [LSP82]. A special consistent failure is a *fail-silent* failure. That is a failure where the system component either delivers a correct service or none at all. For tolerating k component failures, a minimum number of components is required. These are $k + 1$ components for fail-silent components, $2k + 1$ for fail-consistent components, and $3k + 1$ for Byzantine-failing components [PSL80].

The severity of the *consequences* (also called effects) of a failure are also used to classify failures. A failure is called *benign*, if failure costs are in the same order of magnitude as the service delivered. It is *malign* (or catastrophic), if the consequences of failure jeopardize human life and/or are incommensurably greater than the benefit provided by the service delivery in the absence of a failure.

The *oftenness* of the occurrence of a single failure is used to classify failures. A *permanent* failure occurs once and leads to a stop of service provision of a system. A *transient* failure is a failure where the system continues service after the occurrence of the failure. A frequently occurring transient failure is also called *intermittent* failure [Kop97, p.121].

Error. An *error* is the manifestation of a fault [SS92]. It is that part of the system state which is liable to lead to subsequent failure [Lap92]. This part of the internal system state is considered as incorrect [Kop97]. If there is an error in the system state, then there exists a sequence of actions that

can be executed by the system, which will lead to a system failure unless some corrective measures are employed. Whether an error will cause a failure depends on three main factors [Lap92]:

1. The system composition, and especially the nature of the existing (intentional or unintentional) redundancy.
2. The system activity may cause an incorrect internal state to be overwritten with a correct state prior to causing a failure.
3. The definition of a failure from the user's viewpoint.

Kopetz distinguishes two types of errors: transient and permanent errors [Kop97]. A *transient* error exists only for a short interval of time and disappears without an explicit repair action. A *permanent* error persists permanently until an explicit repair action removes it. Powell divides errors in two types: value errors and timing errors [Pow92]. A *value error* is an error where the value of service delivered by a system in terms of a sequence of service items does not fall within the set of values specified. A *timing error* occurs whenever a service item is delivered outside its specified set of durations. The ability to detect errors depends either on regular behavior of nodes or on redundant computation [Kop97].

Fault. The cause of an error is a *fault*. The sources of faults are extremely diverse and can be classified by different viewpoints as depicted in Figure 2.3.

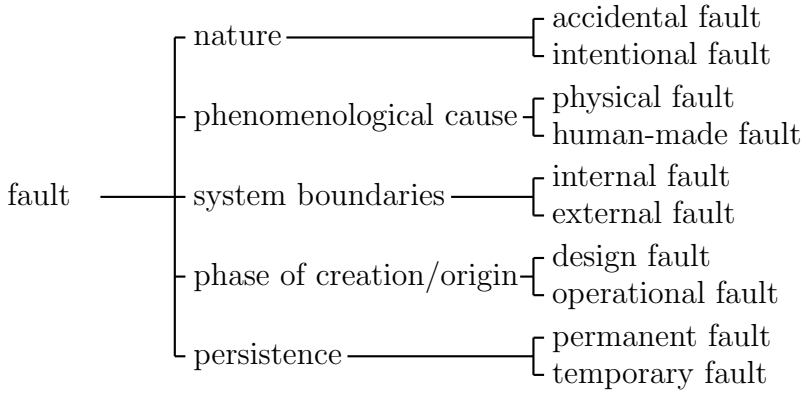


Figure 2.3: Fault classification [Lap95, Kop97].

Using the *nature* of a fault for a classification, a fault can either be *accidental* or *intentional*. An accidental fault appears or is created fortuitously.

An intentional fault is created deliberately. According to Avizienis [Avi78], the *phenomenological cause* of a fault can either be *physical* – that is, a consequence of some adverse physical phenomenon – or *human-made* – that is, due to human imperfection. A classification using *system boundaries* leads to *internal faults* and *external faults*. For internal faults, the cause is a deficiency within the system, while for external faults, the cause is part of the (physical or human) environment of a system. Another point of view for classification of faults is the *phase of creation* (also called origin). A *design fault* has its origin in the incorrect development of a system. An *operational fault* stems from imperfect system operation. And finally, the persistence of a fault can be used for a distinction. A fault can either be *permanent*, that is it persists independently of internal or external conditions; or it can be temporary, that is depending on certain conditions and present only for a limited amount of time.

Fault Pathology. Due to the recursive definition of systems in terms of components, a failure at a given level of decomposition may naturally be interpreted as a fault at the next upper level of decomposition, thus leading to a hierarchical causal chain.

As mentioned above, a fault may lead to an error, which is an incorrect state of a part of a system. This incorrect part of the system state may result in a failure of a system. Systems are composed of components. At a certain level of abstraction, components themselves can be seen as systems. Consequently, a failure can be interpreted as a fault in a larger system, and so on [Lap92]. A similar fault pathology can be constructed for interdependent systems. These fault pathologies lead to a recursive fault-error-failure-model as depicted in Figure 2.4, where an arrow depicts a causal relationship, brackets and 'II' depict system boundaries or abstraction levels.

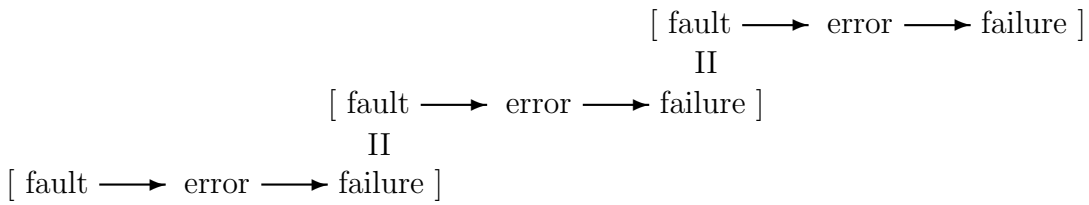


Figure 2.4: Fault-error-failure-model

Fault Hypothesis. For the design of fault-tolerant systems, assumptions concerning all possible faults that may occur and their respective frequency of occurrence must be made. These faults have to be tolerated as long as the

frequency of occurrence is not higher than the stated one. These assumptions determine design decisions, such as the number of necessary replication of components or the used algorithms. The statements about the assumptions that relate to the type and the frequency of faults that the system is supposed to handle are called the *fault hypothesis* [Kop97].

Fault Containment Region. For the design of a fault-tolerant distributed system, assumptions about the consequences of a fault, which is the failing of components, is of relevance. As defined in Section 2.1, a *fault containment region* is a *"set of components that is considered to fail (a) as an atomic unit, and (b) in a statistically independent way with respect to other fault containment regions."* [JKK+01].

Error Confinement Domain. Similarly, an *error confinement domain* (which is also called *error containment region* [KBJ00]) is a component (such as a node or a subsystem of a computer system) that is encapsulated by error detection interfaces in a way that the consequences of an error that can be manifested within this component will not propagate outside this component without being detected [SJ82].

The relationship between error confinement domains and fault containment regions depends on parameters, such as the architecture of a distributed system, the mechanisms for detecting errors and the fault hypothesis. For example, in a subsystem of two connected and mutually checking nodes, where one node can only communicate to other system nodes via the other node and each of the two nodes forms a fault containment region, the two nodes together form an error confinement domain. Similar to this example, Kopetz argues in [Kop02a] that in the Time-Triggered Architecture (see Section 2.3) at least two fault containment regions are necessary to build an error confinement domain for timing failures. One of the two fault containment regions produces (possibly erroneous) messages and one detects and isolates messages that fail in the time domain.

Attributes of Dependability

The attributes of dependability are measures and enable the assessment of the dependability of a system. In detail, the attributes are:

Availability. The availability of a system as a function of time is the probability that the system is operational at a given instant. Availability is typically used as a measure in systems in which service can be delayed or denied for short periods without serious consequences [SS92].

Reliability. The reliability of a system as a function of time is the conditional probability that the system has survived the interval $[0, t]$, given that the system was operational at time $t = 0$. Reliability is used to describe systems in which repair cannot take place (as in satellite computers) or systems in which repair is prohibitively expensive [SS92].

Safety. The safety of a system is the probability that the system will not exhibit a specific undesired behavior throughout a specific period. Safety is a measure for the time to a catastrophic failure.

Security. The security of a system is the dependability with respect to the prevention of unauthorized access and/or handling of information [Lap92]. Security can be defined as the provision of three characteristics: secrecy, integrity, and availability. Secrecy means that the system is protected to prevent unauthorized access and disclosure of state. Integrity of a system means that unauthorized modification of the system and its state is prevented.

Means for Dependability

As depicted in Figure 2.1, the means for dependability can be divided into the two groups *dependability validation* and *dependability procurement*.

Dependability Validation. Means for dependability validation aim at reaching confidence in the system's ability to deliver a service complying with the specification. *Fault removal* and *fault forecasting* are the two means forming dependability validation. Fault removal encompasses methods and techniques to reduce the presence, the number, and the seriousness of faults. Fault forecasting encompasses methods and techniques to estimate the present number, the future incidence, and the consequences of faults.

Dependability Procurement. Means for dependability procurement aim at providing the system with the ability to deliver a service complying with the specification. *Fault prevention* and *fault tolerance* are the two means forming dependability procurement. Fault prevention encompasses methods and techniques to prevent the occurrence or introduction of faults. Fault tolerance accepts that an implemented system will not be perfect and that measures are therefore required to enable the operational system to cope with developing or present faults. To achieve this, methods and techniques must be available that provide a service complying with the specification in spite of faults.

Since this thesis presents a fault-tolerant algorithm, fault tolerance is described in more detail in the next section.

Fault Tolerance

Lee and Anderson describe four phases for achieving fault tolerance in [LA90]. The first three phases together achieve fault tolerance by trying to remove errors from the system state before failures occur. The fourth phase addresses fault tolerance by attempting to eradicate faults from a system before faults lead to errors.

1. *Error detection* aims at detecting an erroneous system state. That is a state that could lead to a system failure, if corrective actions are not performed.

2. *Damage confinement and assessment.* After the detection of an error, the damage of a fault is confined and assessed, because a fault can lead to several errors and errors can propagate before detection. That is, all possible erroneous transitions are identified using known constraints. This information is then used for an estimate of the actual extent of the damage.

3. *Error recovery* aims at eliminating errors from the system state based on the estimate of the first two phases. In this phase, the state that could lead to a failure of the system is substituted by a correct state in order to prevent the occurrence of failures.

4. *Fault treatment and continued service.* The aim of fault treatment is to locate the causes of errors. After the location of a fault, the part of a system that is considered to be faulty (often a component) is repaired or removed from the system in order to prevent impingement of this part on the future operation of the system.

Fault tolerance techniques do not necessarily have to implement all of these phases. Laprie [Lap92] categorizes two primary techniques for achieving fault tolerance. The first category of techniques, called *error processing*, implements the first three of the above mentioned phases. The second category of techniques focuses on the last of the above mentioned phases and is called *fault treatment*.

Error processing can be carried out via three primitives. This is, first, *error detection*, which is basically an implementation of the first of the above mentioned phases. Secondly, *damage assessment*, which implements the second

phase preceded by error detection. And thirdly, *error recovery*, which is an implementation of the first three phases of fault tolerance. The substitution of the erroneous state by a correct state may take on three forms [PS01]:

Backward (or Pessimistic) Recovery [HLMR90]. The transformation of the erroneous state consists of bringing the system back to a state already occupied prior to error occurrence. To achieve this, the state of a system is periodically saved in stable storage. This saved state is called a *checkpoint* and is assumed to be without error. Upon detection of an error, the system restores the current state by the saved state; the system is said to "roll back" to the latest checkpoint and restarted using the system state of this checkpoint.

Forward (or Optimistic) Recovery [PV94, LFA90]. The transformation of the erroneous state is performed by finding a new state, from which the system can operate. For this checkpoints are not necessary. The system is allowed to make progress with the guarantee that the system will return to a consistent state. If arbitrary inconsistencies are allowed, self-stabilization of the system is required for achieving a consistent state after detection of inconsistencies. A detailed description of self-stabilization will be provided below.

Compensation [PS01]. The transformation from an erroneous state into an error-free state is possible, because the erroneous state contains enough redundancy. Adding error detection mechanisms to the component's computational capabilities leads to the notion of *self-checking component*, for the hardware [CS68] or software [YC75]. In a system composed of nodes where each node defines an error confinement domain and computational tasks are replicated at different nodes, error recovery reduces to a switchover of control from a failed node to a non-failed one. Fault masking can be provided by performing error compensation systematically by replicating components and performing voting [BK00]. Then, error detection is not necessarily needed to perform recovery. In order to avoid an undetected decrease in the redundancy available during a component failure, implementations of fault masking usually include error detection. The level of replication for providing error compensation is determined by the fault hypothesis.

According to Laprie [Lap92], the second major technique to achieve fault tolerance is fault treatment. This must begin with fault diagnosis, that is the determination of the causes of errors in terms of localization and nature. The next step is the isolation of faults. *Fault isolation* is the prevention of faults

from being reactivated. This can be done by removing components deemed faulty from the subsequent execution process. If the system service cannot be uphold, a reconfiguration may be envisaged by modifying the system structure so that fault-free components provide an adequate, although degraded, service [PS01].

Kulkarni and Arora [Kul99, AK98a, AK98b] classify fault tolerance into the following three types using a safety and a liveness property: *masking*, *non-masking*, and *fail-safe*. For fault tolerance using masking, the program satisfies its safety and liveness property despite of faults (as long as the fault hypothesis holds). In the case of non-masking, the program need not satisfy its safety properties in case of faults. When faults stop occurring, the system eventually resumes satisfying both its safety and liveness properties. For fault tolerance that implements fail-safe behavior, the system satisfies its safety properties in the presence of faults, but the liveness properties need not hold after faults have occurred.

Self-Stabilization

The notion of self-stabilization was introduced by Dijkstra [Dij82, Dij74]. *"An algorithm is said to be* self-stabilizing *if it converges to a stable 'good' state starting from an arbitrary initial state. Self-stabilization is a good model for algorithms that recover automatically from transient faults – these are faults that disturb operation of the system for some period, and then cease [Rus02]."* This stable 'good' state is also called 'legitimate' [Dij82, Gho93] or 'safe' [GM91] state; a state that is not good is said to be 'illegitimate' or 'unsafe'.

A formal definition of self-stabilization is given by Schneider [Sch93a]. Schneider defines self-stabilization of system S with respect to predicate P and Q over its set of global states. P is intended to identify the correct execution. Q identifies the state from which S stabilizes. A system S where Q is established self-stabilizes to predicate P, if it satisfies the following properties:

- Closure: P is closed under execution of S. That is, once P is established in S, it cannot be falsified.
- Convergence: starting from a global state satisfying Q, S is guaranteed to reach a global state satisfying P within a finite number of state transitions.

This definition shows that a system in general need not recover from any state of the system. Certain preconditions must hold, otherwise the system will not stabilize. A self-stabilizing system as an approach to fault tolerance has

two useful system properties: first, the system need not be initialized (as long as Q is satisfied the system will startup), and, secondly, it can recover from transient failures.

2.1.2 Aspects of Distributed Systems

This section describes several aspects of distributed systems, such as the time aspects, aspects of system composition, and communication aspects.

Synchronous Versus Asynchronous Distributed Systems

Guarantees concerning the timing are used to classify distributed systems either as *synchronous* or *asynchronous*. Asynchronous distributed systems do not require any assumptions concerning the timing behavior of system components for the system to operate correctly. Synchronous distributed systems, on the other hand, require that assumptions concerning the timing behavior of components are met by system components. In more detail, a distributed system is said to be *asynchronous* if there is no fixed upper bound on how long it takes for a message to be delivered or how much time elapses between consecutive steps of a processor [AW98]. In order to guarantee a certain behavior of a distributed system in the time domain, system components must adhere to certain timing rules. A distributed system is classified as *synchronous* if it satisfies the following conditions [HT93]:

Bounded Message Transmission Delay. There is a known upper bound ϵ on message delay. ϵ consists of the time it takes for sending, transporting, and receiving a message over a communication medium.

Bounded Clock Drift. Every node n has a local clock C_n with known bounded rate of drift $\rho \geq 0$ with respect to physical time.

Bounded Processing Time. There are known upper and lower bounds on the time required by a process to execute a processing step.

The asynchronous and the synchronous distributed system models are the two extremes of possible models that are concerned about timing properties. Other models are also studied in literature [DDS87, DLS88, CF99, CF98]. An algorithm designed for an asynchronous distributed system can be used in any other system, because it is independent of any particular timing parameters. Algorithms built on top of synchronous systems can take advantage of assumptions about the timing and allow timeliness guarantees. Generally speaking,

algorithms are easier to implement on top of synchronous systems than on top of asynchronous systems; some algorithms even cannot be implemented on asynchronous systems [CHTC96, FLP85].

Composability

The design of dependable distributed systems requires means for complexity reduction. Composability is such a means for complexity reduction, which addresses the constructive design of large systems out of independently developed pre-validated components [Kop01b]. Composability is the ease of forming a whole by combining parts, where "parts" are the components and the "whole" is the distributed system. A distributed system is said to be *composable* with respect to a specified property, if the integration of an additional component will not invalidate this property once the property has been established at the component level [Kop97, p.34]. For Kopetz, an architecture must adhere to the following four principles, in order to be composable [Kop02b, KS02, Kop00a]:

Independent Development of Nodes. The development of a composable architecture must follow a two-level design philosophy, where the architecture level is defined a priori to the node level. The architecture level is concerned with the development of the precise interface descriptions in the value and the time domain and with the conceptual interface model of the node service. The node level of design then focuses on the interaction with the environment and the development of the node software.

Stability of Prior Services. Services of a node that are validated and exist prior to the integration of this node must not be refuted by the integration of this node.

Performability of the Communication System. This principle is concerned with the resource availability and timeliness of the communication system. The integration of a new node must not disturb the communication patterns and must not lead to a disruption of the timeliness of already integrated nodes.

Replica Determinism. A set of nodes is *replica determinate* [Pol94], if all elements of this set have the same externally visible state and produce the same output messages at points in time that are at least d time units apart, where d is the duration necessary to replace a missing or erroneous message by a correct message.

A common notion of time at different nodes is a requirement for the implementation of the replica determinism and a guaranteed performability of

the communication system, because voting over replicas of a real-time object measured at approximately the same time can be performed and accesses to common resources can be coordinated using time.

The difference between the terms 'scalable' and 'composable' lies in the focus of extension and its services. A system is said to be scalable, if it can be extended. That is new nodes can be integrated, while the system service gets a composite of all node services. A system is said to be composable, if after the integration of nodes the services of all nodes are still available. Consequently, a scalable system must be composable with respect to the properties of the node services [Ste02a].

Broadcast Delivery of Messages

The two extreme types of connectivity of communication system architectures can either be point-to-point or a single shared channel that connects all nodes, such as a bus, a token ring, or a star network architecture with broadcast delivery capability. For nodes, the actual architecture of the communication system may be irrelevant, if the communication system provides a special *network surveillance protocol* [Kim00] (also called *membership protocol* [BP00]) or a guaranteed delivery type despite of failures, such as a fault-tolerant broadcast.

Hadzilacos and Toueg describe different fault-tolerant broadcast types in [HT94]. The weakest type of fault-tolerant broadcast is *reliable broadcast*, which guarantees three properties: first, all correct nodes agree on a set of messages they deliver (*agreement property*); secondly, all message broadcasts by correct nodes are delivered (*validity property*); and, thirdly, no spurious messages are ever delivered (*integrity property*). Another type of fault-tolerant broadcast is *FIFO broadcast*. FIFO broadcast exhibits all properties of reliable broadcast and, in addition, messages are delivered in the order they are broadcast. If messages are also causally ordered, this is called *causal broadcast*. Yet, if messages are not in causal relationship, the order of delivery may be different for different nodes. *Atomic broadcast* guarantees the delivery of all messages in the same order. Further combinations of the listed broadcast types are possible and described in detail in [HT94].

2.1.3 Types of Distributed Systems

The following paragraphs describe different types of distributed systems, namely real-time systems, multi-cluster systems, and embedded systems. While some of these systems do not necessarily have to be designed in a distributed manner, this thesis describes these system types in the context of distributed systems, because the focus will be put on properties related to distribution.

Real-Time Systems

According to Kopetz [Kop97, p.2], *"a real-time computer system is a computer system in which the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical instant at which these results are produced."* In this thesis, a *real-time system* comprises a *real-time computer system* and an *environment*, which itself consists of a *controlled object* and a *human operator*. While real-time computer systems are not generally distributed systems, we consider only real-time computer systems that have a distributed system architecture. A distributed solution has several advantages over a central solution, such as dependability arguments, scalability, and physical installation [Kop97]. In this thesis, the terms 'real-time computer system' and 'real-time system' are used interchangeably.

This definition of real-time systems is in contrast to known other definitions of real-time systems where systems do not have an awareness of time and where computations do not take any time, as Lee states in [Lee99]: *"Time has been systematically removed from theories of computation, since it is an annoying property that computations take time. Pure computation does not take time, and has nothing to do with time. It is hard to overemphasize how deeply rooted this is in our culture. So called real-time operating systems have so little to go on that they often reduce the characterization of a component (a process) to a single number, its priority."* Since real-time systems must be aware of the progression of time, they can only be of synchronous and not asynchronous nature.

The computation of a real-time computer system is activated by stimuli of its environment. The latest instant at which the real-time computer system can produce the result of the computation is called the *deadline*. A real-time computer system that produces a result after a deadline is said to have *missed* its deadline. If a result has (degraded) utility after its deadline, the deadline is said to be *soft*, otherwise *firm*. If the miss of a firm deadline can result in a catastrophe, this deadline is called more specifically *hard* deadline. A computer system that must meet at least one hard deadline is called a *hard* real-time computer system [SR88] or a *safety-critical* real-time computer system [Kop97].

Real-time systems can be classified according to the behavior of the system in case of failures. If a fail-safe state can be reached in case of a system failure, the real-time system is said to be *fail-safe*. In real-time systems, where a fail-safe state does not exist (such as in a control system aboard of an airplane), the real-time computer system must provide a minimum level of a service despite of failure. Such a system is called *fail-operational*. It is important to realize that the classification of a system as fail-safe or fail-operational is determined by the environment and not the real-time computer system [Kop97, p.14].

Time-Triggered Systems. A special class of distributed real-time systems are *time-triggered systems*. In "pure" time-triggered systems, the only stimulus of system activities is the progression of time. These systems are efficient if regular patterns for communication and computational activities are required. The activation patterns of activities are agreed and known to all nodes. Examples for time-triggered systems are the Time-Triggered Architecture (TTA) [KB02], SPIDER [Min00], SAFEbus [ARI93, HD93], or FlexRay [MHB+01].

Multi-Cluster Systems

The principle of Alexander [Ale64], *"The ultimate object of design is form."*, and the guideline of Vitruvius [Vit96], *"Form follows function."*, guide the design of systems as multi-cluster systems. A *cluster* is a group of nodes that directly communicate and pursue a common computational goal. *Multi-cluster systems* consist of a number of clusters where nodes of the same cluster have a high inner functional and temporal coupling. Clusters are, in turn, interconnected. The coupling between clusters is, however, much looser than the coupling within the single clusters.

The reasons for designing systems in the form of multi-cluster systems are manifold. Multi-cluster systems are one approach to complexity management of large distributed systems and to solve limitations of communication bandwidth. Multi-cluster systems can also result from the gradual development of systems.

Rechtin lists heuristics for partitioning of systems in [Rec91], which lead to a well structured multi-cluster system:

- *"Except for good and sufficient reasons, functional and physical structure should match. [...]*
- *In partitioning, choose the elements so that they are as independent as possible, that is, elements with low external complexity and high internal complexity. [Ale64] [...]*
- *In partitioning a distributed system, choose a configuration in which local activity is high speed and global activity is slow change. [Cou85] [...]*
- *In partitioning a system into subsystems, choose a configuration with minimal communication between the subsystems. (For example, aerospace, communication network, and software systems.) [...]*
- *Don't partition by slicing through regions where high rates of information exchange are required. (For example, computers.) [...]"*

Embedded Systems

Embedded computer systems (for short *embedded systems*) are computer systems embedded in devices designed for control and monitoring application. The term 'embedded' denotes that an embedded system is part of a larger system, which is called an *intelligent product* [Kop97]. An intelligent product consists of a mechanical subsystem, the controlling embedded computer, and, sometimes, a man-machine interface. An embedded system is in close interaction with its environment, which is often a physical or mechanical process.

Embedded systems and their system development process are characterized by [Pas02, Kop97, BCE+01]

- a missing or sharply reduced importance of the *user interface*;
- *direct interaction of the software with hardware peripherals*, where multiple inputs and outputs demand a high degree of concurrency;
- *guaranteed response time* due to close interaction with environment and employment for control application;
- *greater autonomy needs compared to desktop computers*, which follows from the first two above mentioned characteristics and the design of embedded systems for stand-alone operation;
- a high degree of *dependability* due to its autonomy needs, the expectation of longevity, the near impossibility to modify embedded system software after release;
- a provision of an excellent *diagnostic and maintenance interface* if required;
- constraints in *limited hardware resources* determine design and implementation decisions (for example, power-aware processing due to battery operation; software functionality determined by read-only memory size; low computational power, small working memory compared to desktop computers and communication bandwidth limitations; the need for adequate heat dissipation);
- need for *composable system components* for networked embedded systems structure, because inter-operability is a key concern;
- *mass production* and *minimization of mechanical subsystems* due to deployment of embedded systems in large volume application.

2.2 Clocks, Time, and Clock Synchronization

This chapter introduces the reader to the concepts and terms used for clock synchronization and available time standards. These concepts and terms have been introduced in [KO87, FC97, Kop97, Sch88].

2.2.1 Concepts of Clocks

In distributed control systems, the order of occurrences of events can be established, if a common notion of time is available. Time is established using different clocks. This section describes the different types of clocks used in this thesis.

Reference Clock. In this work the concept of a reference clock is introduced due to the following reason. Real time can be depicted as time line (if relativistic effects are disregarded) and events occur at points on this time line. Let $\mathcal{T}$ denote the set of all real-time values and t the point in $\mathcal{T}$ of the occurrence of an event. Real-time is a dense time base. That is, let t_1 and t_2 denote the time of occurrences of event 1 and 2, respectively, then there can always be another event occurring at t_3, say event 3. t_3 can be between t_1 and t_2, no matter how close t_1 and t_2 are. Consequently, the granularity of real time is infinitesimal and a model using real time must use real values and floating point operations for modeling event occurrences and temporally ordering these events. This is a resource-consuming approach.

Reference clock time, denoted $\mathcal{RT}$, is a granular representation of real time, where the granularity of $\mathcal{RT}$, denoted $g_{\mathcal{RT}}$, is so small that digitalization errors that occur when timestamping events can be neglected in the analysis of the model. The reference clock time is defined by a sequence of ticks $\mathcal{RT}_i$, $i \in \{0, 1, 2, 3, \dots\}$, everywhere accessible, and is in perfect synchrony with real time. That is $\mathcal{RT}_i \in \mathcal{T}$ and for any i and any point in real time t, $\mathcal{RT}_i \leq t < \mathcal{RT}_{i+1}$ is valid. The occurrence of an event can be timestamped using a clock. $clock(event)$ denotes the timestamp generated by the use of clock *clock*. Consequently, the use of the reference clock time to timestamp event e_1 is denoted $\mathcal{RT}(e_1)$.

The concept of a reference clock is closely related to the implementation of clocks in computer systems. It allows the representation of time using integer numbers and for ordering of timestamped events simple integer arithmetic suffices. The concept of the reference clock allows simple models and simple definitions of clock parameters and, thus, alleviates implementation, formal verification and validation of these models.

Hardware Clocks. A hardware clock typically consists of an oscillator and a counting register that is periodically incremented by periodically generated events of an oscillator. The periodic event is called the tick. The time of a correct hardware clock, which is the contents of the counting register, will monotonically increase as long as the counting register does not overflow. The duration between two consecutive ticks is the granularity of the clock. This granularity leads to a digitalization error in time measurement. The time of hardware clock i is denoted $\mathcal{HT}^i$. Ticks are numbered using natural numbers. $\mathcal{HT}^i_k$ denotes tick number k of hardware clock i. Using this notion, the duration of the granularity of tick k of hardware clock i equals $\mathcal{RT}(\mathcal{HT}^i_{k+1}) - \mathcal{RT}(\mathcal{HT}^i_k)$.

Drift Rate of a Clock. Real clocks are not perfect. That is, they drift with respect to the reference clock. This drift is due to impressions of the oscillator, changes in the abient temperature and other environmental conditions change, and aging of the crystal. A measure for the imperfect behavior of a clock is the drift rate of a clock. If n_k denotes the nominal number of reference clock ticks of clock tick k, the drift rate of clock i for tick k is defined as

$$\rho^i_k = \left| \frac{\mathcal{RT}(\mathcal{HT}^i_{k+1}) - \mathcal{RT}(\mathcal{HT}^i_k)}{n_k} - 1 \right| \tag{2.1}$$

The drift rate of clock i, denoted ρ^i, is the maximum drift value of ρ^i_k in an interval of interest. The drift rate of a perfect clock equals 0. For specified environmental parameters, the drift rate of a clock is bounded by the maximum drift rate, denoted $\bar{\rho}^i$.

Schwabl observed in [Sch88] that the drift rate consists of a *systematic* and a *stochastic* part. The categorization of the stochastic and the systematic error is done from a viewpoint of the observation length and the changes in this period. In a short observation period (seconds to minutes), the systematic error is constant and the value of the stochastic error changes. Yet, for long observation periods, the systematic error changes as does the stochastic error. In this thesis, ρ^i_{sys} denotes the systematic part of the drift rate of clock i and ρ^i_{stoch} the stochastic part. The systematic part of the maximum drift rate of clocks using commercially available quartzes is in the order of $5 \cdot 10^{-5} \frac{s}{s}$ while the stochastic part is in the order of $10^{-7} \frac{s}{s}$. Consequently, the systematic part is at least an order of magnitude larger. The stochastic part of the drift rate originates from changes in the environmental conditions and stochastic processes in the quartz crystals. The systematic error from the nominal frequency stems from the manufacturing process and slow aging of quartz crystals.

Virtual Clocks. The concept of virtual clocks is introduced for two reasons. First, a virtual clock abstracts from hardware-dependent parameters, such as

the frequency of the oscillator. Secondly, for the synchronization of clock times, the frequency of ticking and/or the contents of the counting register must be changed. However, it is expensive to change the frequency of ticking of hardware clocks by changing the frequency of the oscillator.

Virtual clocks work in the following way: an integer number of hardware clock ticks comprise one virtual clock tick. For a correction of the clock frequency (acceleration or deceleration) of a virtual clock, the number of hardware clock ticks per virtual clock tick is changed using an adjustment value, denoted $H^i(t)$. The initial clock state of a virtual clock can also be set using the adjustment value. This adaption of the number of hardware clock ticks per virtual clock tick can be used to synchronize the time of virtual clocks. Figure 2.5 depicts a simple concept of a virtual clock. In this figure, the value of the virtual clock time is incremented every g hardware clock ticks, where g is the nominal granularity of the virtual clock measured in hardware clock ticks. In Figure 2.5, the register value of the hardware clock (depicted as rectangle labelled "hardware clock") is monotonically increasing due to an oscillator. Whenever the hardware clock value modulo g equals zero, the virtual clock is increased by one tick. In addition, an adjustment value can be added to the virtual clock time, as depicted by the rectangle labelled $H^i(t)$. The adjustment value can change over time. $\mathcal{VT}^i$ denotes the time of virtual clock i, that is all ticks of this clock, and $\mathcal{VT}^i_k$ denotes tick k of this clock. The value of the virtual clock (which is the contents of the register depicted as rectangle labelled "virtual clock") can be accessed and read.

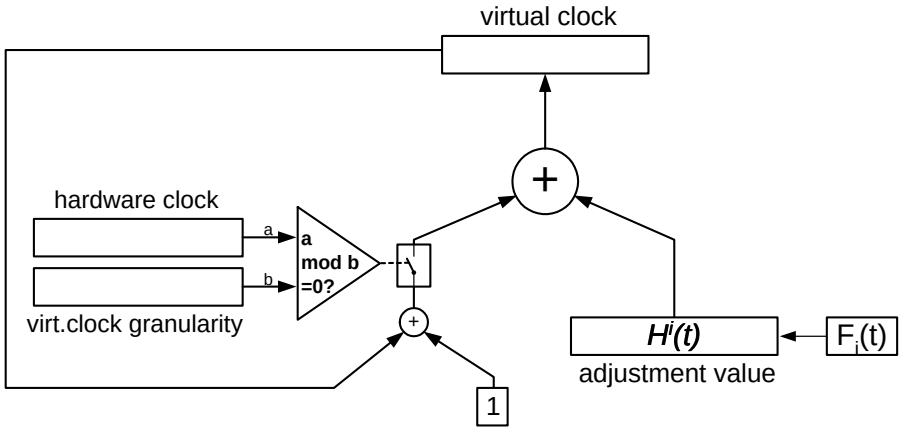


Figure 2.5: Concept of a virtual clock, which comprises a hardware clock and an adjustment value

The term "virtual" in the notation "virtual clock" stems from the synchronization process of clocks. Instead of synchronizing hardware clock times, the times of virtual clocks are synchronized (see Section 2.2.2). The term "virtual" may mislead to the conclusion that the time of a virtual clock cannot

be directly accessed or read. Yet, the time of a virtual clock *can* be directly accessed or read by a node. The term "global time" is used for referring to the synchronized times of several virtual clocks (see Section 2.2.4 for details). The global time of these clocks is an *abstract notion* and cannot be directly read by nodes. Instead, a node can only access its virtual clock time that forms the global time – together with other virtual clock times. Consequently, a node can only obtain an approximation of the global time. This is why a virtual clock time is also called an *approximation of the global time at a node* [KO87].

Relational Clocks. Reference clocks are ideal representations of real time. In reality, even clocks that are used as time reference will not ideally represent real time. To accommodate this aspect in this thesis, the *notion* of a relational clock is introduced. A *relational clock* is a clock used as reference for externally synchronizing clocks (see Section 2.2.2 for an explantation of external clock synchronization). A relational clock is a notion for a virtual clock that is used as a reference. Relational clocks will *not* be in perfect synchrony with real time. That is, a relational clock will have a maximum drift rate different from 0 and the offset between reference clock time and relational clock time will also be different from 0. $\mathcal{RCT}^j$ denotes the time of relational clock j and, similarly to the use of notion of other clocks, $\mathcal{RCT}^j(e_1)$ denotes the timestamp of event e_1 using relational clock j.

Offset. The offset at a given time between two clocks is the difference between the two clock times. Using the reference clock, the offset between the two virtual clocks i and j with the same granularity for tick k at both clocks can be defined as follows:

$$O_k^{i,j} = \left|\mathcal{RT}(\mathcal{VT}^i{}_k) - \mathcal{RT}(\mathcal{VT}^j{}_k)\right| \tag{2.2}$$

The offset between a virtual clock time j and a relational clock time j at time t is denoted as Ω_t^j.

2.2.2 Clock Synchronization

Since any clock drifts, the clock times of an ensemble of clocks will drift apart, if they are not periodically re-synchronized with respect to each other[1]. Clock synchronization is concerned with bringing the time of clocks in close relation with respect to each other. A measure for the quality of synchronization is the *precision*.

[1] "Re-synchronize clocks with respect to each other" means "to bring the different times of the clocks closer together".

Precision. Given an ensemble of clocks, the maximum offset between any two clocks of this ensemble in an interval of interest is called the precision. For an ensemble of virtual clocks with the same granularity, where the virtual clocks are numbered from 1 to N, the precision at tick k is defined as

$$\Pi_k = \max_{1 \leq i,j \leq N} (O_k^{i,j}) \tag{2.3}$$

The maximum of Π_k over an interval of interest is called the precision of an ensemble of clocks, denoted Π. This precision Π is expressed in the number of ticks of the reference clock.

Internal Clock Synchronization. The process of mutual re-synchronization of an ensemble of clocks to maintain a bounded precision is called *internal clock synchronization*. Internal clock synchronization does not necessarily mean synchronization to real time, as all clocks of an ensemble can drift. Internal clock synchronization is defined *optimal* by Srikanth and Toueg [ST87], if the drift rate of the synchronized clock time of an ensemble of clocks (with respect to real time) is smaller than or equal to the drift rate of the largest drift rate of the ensemble of clocks.

External Clock Synchronization. If an ensemble of clocks synchronizes their clock times to distinguished clocks that are not part of this ensemble, this is called *external clock synchronization*. A quality measure for this kind of synchronization is the accuracy.

Accuracy. Given a clock, the accuracy of this clock with respect to a reference clock is the maximum offset between these two clocks in an interval of interest. For virtual clock i, the accuracy with respect to the reference clock time is defined as

$$\alpha^{\mathcal{VT}^i,\mathcal{RT}} = \max_{\forall k} \left(\left| \mathcal{RT}(\mathcal{VT}_k^i) - \mathcal{RT}_l \right| \right) \tag{2.4}$$

where the reference clock tick l is the corresponding tick of the reference clock time of tick k. The accuracy of an ensemble of clocks is defined as the maximum value of the accuracy of all clocks of an ensemble. Please note that this definition of accuracy is in contrast to other known definitions of accuracy in the area of clock synchronization [Sch87, ST87]. In the cited papers, the accuracy defines the maximum drift rate that any synchronizing clock has.

The accuracy of the reference clock time with respect to real time is *not* zero. Due to the definition of the reference clock time, the accuracy of the

reference clock time with respect to real time, denoted $\alpha^{\mathcal{RT},\mathcal{T}}$, is

$$\alpha^{\mathcal{RT},\mathcal{T}} = g_{\mathcal{RT}} \tag{2.5}$$

where $g_{\mathcal{RT}}$ is the granularity of the reference clock.

The accuracy of virtual clock i with respect to relational clock j with the same nominal granularity is defined as:

$$\alpha^{\mathcal{VT}^i,\mathcal{RCT}^j} = \max_{\forall k} \left(\left| \mathcal{RT}(\mathcal{VT}^i_k) - \mathcal{RT}(\mathcal{RCT}^j_k) \right| \right) \tag{2.6}$$

In this thesis, the accuracy of a virtual clock with respect to a relational clock is defined, because it is a measure that enables the evaluation of the quality of a clock synchronization algorithm using relational clocks as references.

External clock synchronization is seen as synchronization to one or more clocks that are synchronized to real time with a synchronization quality (in terms of accuracy values) that is at least by a factor of ten better than the aimed synchronization quality of the externally synchronizing clock(s).

Multi-Cluster Clock Synchronization. This thesis focuses on the synchronization of clocks of multi-cluster systems, which we call *multi-cluster clock synchronization*. Multi-cluster clock synchronization can be seen as external clock synchronization from a point of view of one cluster with the following distinct differences compared to known external clock synchronization:

- The time of the relational clocks used as reference does not have to be synchronized to real time.
- The drift rate of relational clocks is likely to be different from 0 and can be in the order of the synchronizing clocks. This is in contrast to known external clock synchronization algorithms where the drift rate of clocks used as references is assumed to equal zero or is negligibly small compared to the synchronization requirements [Pat94].

Clock Synchronization Versus Synchronization of Clock Rates. Clock synchronization (i.e. synchronization of the times of clocks) is not the same as clock rate synchronization [Lis91]. Clock synchronization does *not* mean only synchronizing the rates of clocks, because synchronizing rates of clocks leaves the initial state of a clock open. Synchronizing clock times, however, automatically implies synchronization of clock rates. Synchronizing rates is computationally less intensive compared to synchronizing times and may suffice for certain applications [Lis91].

State Correction Versus Rate Correction. In order to synchronize clock times, the clock times must be corrected. As stated above, clock synchronization uses virtual clock times. There are two different approaches to correcting virtual clock times: state correction and rate correction. For *state correction*, a node applies a computed correction value at once and immediately after calculation. For *rate correction*, the rate of the clock is accelerated or decelerated by changing the number of hardware clock ticks per virtual clock tick over an interval. The sum of the number of hardware clock time changes in the interval equals the computed correction value.

Table A.1 in Appendix A gives an overview of the most important terms of this thesis.

Principle of Operation of Clock Synchronization

As defined above, the goal of clock synchronization is to bring or maintain the clock times of the synchronizing clocks in close relation to each other or to the time of relational clocks. In order to achieve this goal, each node cyclically performs the following three phases:

Phase 1: Collection of clock time values. In phase 1, a node collects information about the clock time values of other clocks actively participating in the synchronization. This can either be the difference to its own clock time or the actual clock time value.

Phase 2: Calculation of correction value. After performing phase 1, each node computes a correction value using (some or all) of the collected clock time values in phase 2. The computation is performed using the convergence function, which should bring clock times closer together.

Phase 3: Clock correction. In phase 3, a node corrects its clock time using the calculated correction value of phase 2. As described in Section 2.2.1, all implemented clock synchronization algorithms use the concept of virtual clocks. This allows the simple adaption of the clock times.

Schneider argues in [Sch87] that basically all internal clock synchronization algorithms are solutions to the following three subproblems:

Re-synchronization. What is the event that causes the re-synchronization of clocks?

Remote clock time readings. How does a clock obtain the clock time values of other clocks?

Convergence function. How are the correction values of different clocks computed in order to keep clocks synchronized within a bounded interval (the precision) while preserving the monotonicity of the clock times and keeping the drift of the clocks better than or equal to the hardware clock drifts.

Schneider's argument is also valid for external clock synchronization algorithm.

Classifications of Clock Synchronization Algorithms

The following paragraphs classify clock synchronization algorithms

Internal and External Clock Synchronization Algorithms. Clock synchronization algorithms can be classified by the clocks used as reference clocks. If the clocks used as reference are not part of the synchronizing ensemble this is commonly called *external clock synchronization*. Examples of external clock synchronization algorithms are [CF95, MO83]. If the clocks used as reference are also synchronizing clocks, this is called *internal clock synchronization*. There are a number of different internal clock synchronization algorithms protocols, which all basically differ by the different approaches to the three subproblems described by Schneider [Sch87]. Examples for internal clock synchronization algorithms are [BD87, FC95, HSSD84, KO87, LM84, LM85, LL84a, MS85, ST87, RSB90, WF00]. Fetzer and Cristian combined the internal and external clock synchronization [FC97, CF95].

Deterministic, Probabilistic, and Statistical Clock Synchronization Algorithms. Internal clock synchronization algorithms can further be classified by assumptions about the transmission delay of the communication system in *deterministic*, *probabilistic*, and *statistical* clock synchronization algorithms [AP97].

- Deterministic clock synchronization algorithms assume that an upper bound on transmission delay exists. If this assumption holds, a certain quality level of clock synchronization (in terms of a guaranteed precision) can be guaranteed.
- Probabilistic and statistical clock synchronization algorithms do not assume a strict upper bound on the transmission delay. Instead, statistical clock synchronization algorithms assume that the first and second moment of the distribution of the transmission delay and – sometimes – that

the distribution is known. As a consequence, a node does not know the precision of the clock synchronization at a given time. Probabilistic clock synchronization algorithms make no assumption about the distribution of the transmission delay. Instead it is assumed that with probability p, $p < 1$, the transmission delay will be smaller than a guaranteed constant maximum transmission delay. As a consequence all clocks know whether they are synchronized or not at any time [Cri89].

A performance analysis of the different types of algorithms can be found in [AP98].

Hardware Versus Software Implementation Clock synchronization algorithms that are implemented in hardware and use specialized hardware components, such as [KR87, KO87], achieve tight synchronization. On the other side, implementations in software, such as [Cri89, FC95, HSSD84, ST87, LL84a, LM85, MS85, PB95, VRC97], do not achieve synchronization as tight as hardware algorithms, but use commercial-off-the-shelf components.

Related Work on Clock Synchronization Algorithms

Halpern et alii [HSSD84] present a fault-tolerant clock synchronization algorithm that works in arbitrary networks. Approaches to clock synchronization are refined for the employment in large distributed systems [Mar84, SS97, Sch00, VRC97]. Rushby and von Henke have formally verified clock synchronization algorithms [RvH89]. Schedl simulated several clock synchronization algorithms [Sch96].

Most of the presented clock synchronization algorithms use replication as fault tolerance strategy. Dolev, Welch, and Papatriantafilou present approaches that achieve fault tolerance using self-stabilization [DW95, Dol97, PT94]. Analyses of self-stabilizing clock synchronization algorithms can be found in [LZM90, CL94].

Measurement and control applications are increasingly using distributed system technologies such as network communication, local computing, and distributed objects. As these measurement and control applications base on distributed embedded systems, clock synchronization has become an important area for standardization. In 2001, the IEEE standard organization has started a standardization process for clock synchronization in measurement and control applications. In 2002, the standardization committee has published a draft standard [IEE02] for clock synchronization, called "Precision Time Protocol" (IEEE P1588). This standard addresses the needs of measurement and control

systems: microsecond to sub-microsecond accuracy; administration free; and most importantly, accessible for both high-end devices and low-cost, low-end devices. This standard decouples the application and communication task by introducing an application and a communication layer and an isolation layer that isolates application activities from communication activities and provides time provision services. The communication layer of the protocol requires all communicating nodes to follow a time-division multiple access (TDMA) strategy for the communication medium in order to achieve low latency jitter. The TDMA strategy is achieved by implementing a master/slave protocol [Jen02].

2.2.3 Time Standards and Sources

Time standards are an agreed origin and representation of time. This section describes the two time standards *Internal Atomic Time* and *Universal Time Coordinated*. Clocks that are synchronized to time standards are potential sources for references for external clock synchronization. That is, they can be used as relational clocks. We call systems that provide clocks that are synchronized to time standards *time sources*.

Time Standards

International Atomic Time. In 1967, the Bureau International de l'Heure (BIH) specified the *Temps Atomique International* (TAI) in order to provide a time standard that can be produced in a laboratory, but is in agreement with the second derived from astronomical observations. The TAI defines the second as the duration of 9192631770 periods of the radiation of a specified transition of the cesium atom 133. The TAI is a strictly monotonic timescale (also called a *chronoscopic* time scale). That is a timescale without any discontinuities.

Universal Time Coordinated. The *Universal Time Coordinated* (UTC) is a time standard that is in close relation to the time derived from astronomic observations of the rotation of the earth relative to the sun. UTC is a time standard used for business relations and is a basis for widely used synchronized clocks, such as wall clocks. There is an offset between TAI and UTC due to the deceleration of the rotation speed of the earth, which is adjusted in second intervals by the BIH whenever necessary. UTC time is always kept within ± 0.9 seconds of the time derived from astronomic observations (including polar wander effect corrections) by the insertion of an extra second (positive leap second) as needed. While it is theoretically possible to have to remove an extra second (negative leap second), it has not happened so far. From the last statements it can be concluded that UTC is not free of discontinuities.

Time Sources

Global Positioning System. The U. S. Department of Defense funds, controls, and operates the *Global Positioning System* (GPS). GPS is a dual-use, satellite-based system that provides accurate location and timing data worldwide. Provision is done via specially coded satellite signals that can be processed in a GPS receiver, enabling the receiver to compute position, velocity, and time. The time at GPS satellite is synchronized to UTC time with an accuracy that is better than 15 ns (there is an offset between GPS time and UTC to provide monotonicity). The accuracy of the timing information of commercially available GPS receivers using special measurement methods is about 50 ns. GPS is an accepted and widely used time source for military and civil application [LAK99, HS97, Dan97].

Global Navigation Satellite System. The *GLObal NAvigation Satellite System* (GLONASS) is – similar to GPS – a dual-use satellite-based navigation system that enables global-wide positioning, velocity measuring, and timing information. The GLONASS system is managed by the Russian Space Forces for the Russian Federation Government. The GLONASS system provides two types of navigation signals with different precision level. The time of GLONASS satellites is synchronized to UTC (with a constant offset to provide chronoscopic behavior) with approximately 15 ns [Leb98].

Galileo. Galileo is the planned European satellite navigation and time transferring system. The European Union launched the GALILEO project due to the following concerns regarding GPS and to a degree GLONASS [Kha01]: both systems are under the unilateral control of a foreign national defence authority, the absence of guarantees of service, priority on military needs, intentionally degraded use to civil needs, poor availability in urban areas, and unpredictable gaps in coverage. Galileo should overcome these concerns and become fully operational in 2008.

Radio-Controlled Clocks. In several regions of the world, time information is broadcast via radio services with a UTC timecode modulation (examples are DCF77, WWVH, WWV, and HBG). Timecode receivers enable the reception of a time signal with an accuracy with respect to UTC in the order of 10 ms. Receivers, however, are subject to occasional gross errors due to propagation and equipment failures [LAK99, Sch88, Mil91]. Terrestrial radio stations allow for accuracies of timing information in the range of 50 ms up to 10 μs [Lic97].

2.2.4 Time Aspects from an Application-Specific View

Time can serve different aspects for an application. For a real-time control application, time must have properties such as chronoscopic behavior. For applications that use time as reference for users for synchronizing their wall clock and watches, time must be synchronized to UTC and chronoscopic behavior is thus impossible. This section describes different aspects of time.

Optimal Representation of Time for Real-Time Systems

For time in real-time systems, a representation should be chosen that has the following properties:

- no overflow occurs in system life time
- the precision of the time is smaller than the granularity of the representation
- chronoscopic behavior

The time should be accessible via reliable sources world-wide and the jitter at receivers with respect to the reference time should be low. The GPS time is a continuous representation of time with high availability and low jitter. Yet, GPS time overflows every 19.6 years because the GPS week is represented by a 10 bit value. This shortcoming of GPS is overcome by the time format [OMG02] described in Figure 2.6. This representation guarantees no overflow during the lifetime of the product (horizon of more than 10 000 years) and the full second can be easily accessed due to binary representation of a second overflow. The granularity of this time representation is 59.6 ns, which is in the order of the accuracy of common GPS receivers. Using this time representation and a satellite-based time provision system enables the above mentioned requirements.

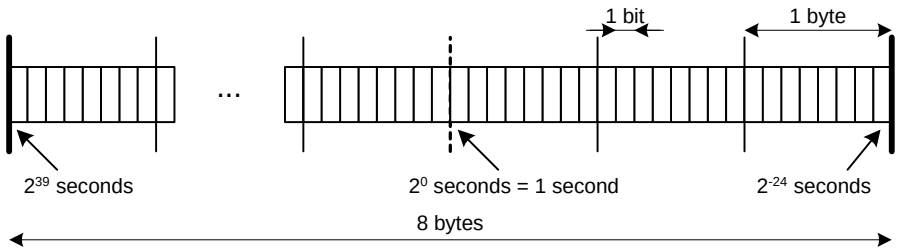


Figure 2.6: Optimal representation of time in real-time systems [OMG02]

Global Time.

The set of time values of an ensemble of clocks that are synchronized to each other with precision Π are called the *global time* of this ensemble. The global time of an ensemble is represented by the local time of each clock at each of the clocks. This local time of a clock is an approximation of the global time [KO87]. The notion of global time is introduced, because it is impossible to perfectly synchronize[2] clocks [LL84b] due to uncertainties and properties of the communication systems (such as the latency jitter and bounded transmission speed of the communication system). This impossibility result has far-reaching consequences, which will be explained in the next paragraphs.

Reasonable Time

A *"global time is called* reasonable*, if all local implementations of the global time satisfy the condition $g > \Pi$[,] the reasonableness condition for the global granularity g" [Kop97, p.52].*

Sparse Time

A model of time is called *dense time*, if time progresses along a *dense* timeline and events can occur at any instant on this time. In a sparse-time model, the time is partitioned into alternating durations of activity and silence as depicted in Figure 2.7.

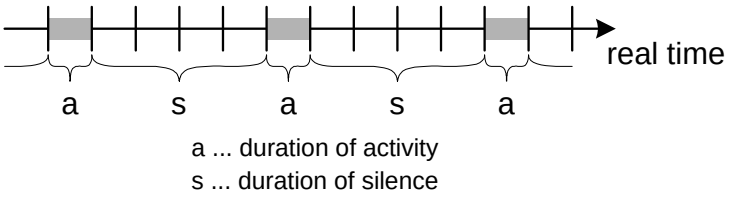


Figure 2.7: Sparse time [KO02]

In a distributed system it is impossible to order events consistently on the basis of their timestamps – if these timestamps are based on a dense time – due to the impossibility result [LL84b]. Timestamps of events based on a sparse time can be ordered, if events with timestamps in the same activity interval are considered to happen at the same time. Events can be totally ordered, if events are timestamped using a sparse time with the following properties: first, the interval of activity is smaller or equal to the precision of the clock synchronization in a distributed system and, secondly, the duration of silence is at least four times larger than the duration of activity [Kop97, KO02].

[2]Perfectly synchronizing clocks means synchronizing clocks with a precision equal to 0.

2.3 Time-Triggered Architecture

This section describes the Time-Triggered Architecture (TTA) [KB02] as an example for a fault-tolerant embedded system architecture, because it is used as evaluation architecture in this thesis. Other examples are SPIDER [Min00], SAFEbus [ARI93, HD93], or FlexRay [MHB+01].

The Time-Triggered Architecture (TTA) is a computer architecture designed to support the requirements of fault-tolerant real-time systems. The TTA provides the following services to the application at the architecture level [Kop02a]:

- A consistent distributed communication and computing platform with prompt error detection if consistency is lost by a failure that can be detected at the architecture level.
- A fault-tolerant global time base of known precision at all nodes that support a sparse time base.
- mechanisms for the precise operational specification of the interfaces among the nodes in the value and time domain.
- Error containment such that single arbitrary node failures can be tolerated.
- Mechanisms that support the transparent implementation of fault tolerance.

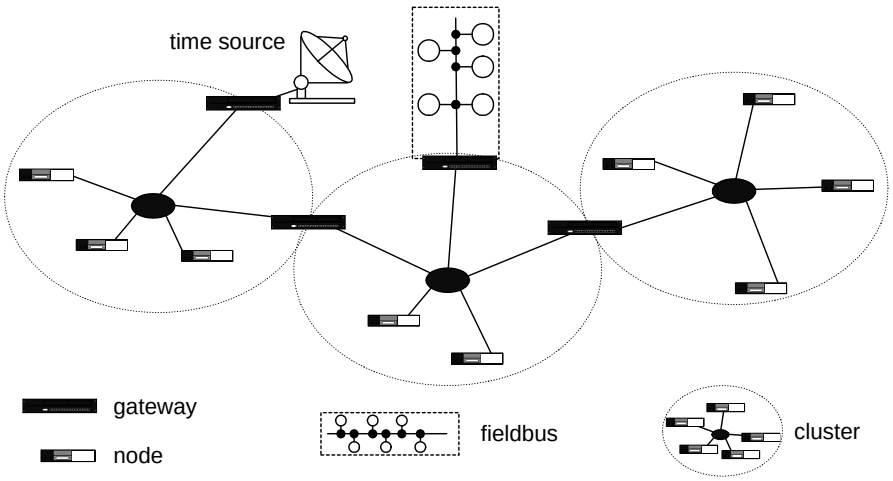


Figure 2.8: Time-Triggered Architecture system

Figure 2.8 shows a TTA system with three clusters. Nodes in a TTA cluster can directly communicate via broadcast messages via the time-triggered communication protocol *TTP/C* [Kop97, Kop99]. Clusters may be connected via gateway nodes to allow communication between nodes of different clusters, that is *inter-cluster communication*. Nodes within a cluster are provided with a time base of known precision, while the synchronization of global times of different clusters is the topic of this thesis. Nodes may be connected to an external time source, also an aspect of this thesis. Gateway nodes may also connect a cluster with a fieldbus for exchanging input/ouput data from/to the environment. This can be a time-triggered sensor bus, such as TTP/A [KHE01]. In order to achieve replica determinism [Pol94], nodes must perform an agreement protocol on the input data.

TTP/C

The next paragraphs describe the communication protocol *TTP/C* of the TTA in more detail. *TTP/C* is designed to tolerate arbitrary single faults. To tolerate single failures of the communication medium, *TTP/C* provides a communications medium consisting of two replicated channels. For a broadcast to be correct, data transmission must be successful on at least one of the replicated channels.

Internal Clock Synchronization and Media Access in *TTP/C*. Media access in *TTP/C* is controlled by a conflict-free TDMA (time-division multiple-access) strategy, which ensures a bounded communication delay. The TDMA scheme divides time into slots of not necessarily equal length and assigns each node a unique slot where only this node is allowed to send. After all nodes have sent, the access pattern is repeated. A single repetition is referred to as *TDMA round* (Figure 2.9).

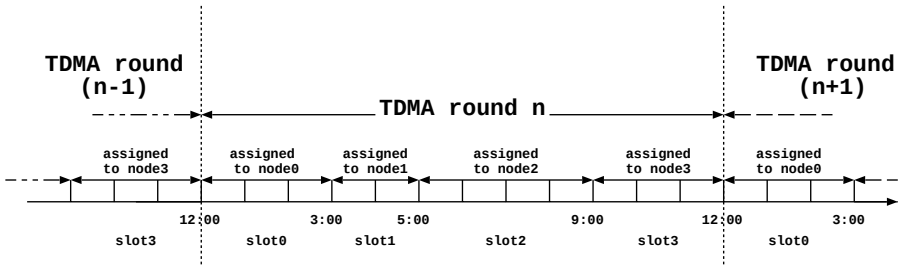


Figure 2.9: TDMA round (four nodes)

Communication media access is time-triggered. An internal clock synchronization algorithm, the Fault-Tolerant Average (FTA) algorithm [Kop97, p.62],

builds the *global cluster time*. Each node computes its time representation using its own clock and the differences between the expected arrival times and the actual arrival times of messages from other nodes. These differences represent the differences of the clock values of other nodes $\mathcal{VT}^i$, $1 \leq i \leq N, i \neq k$, and the value of a node's own clock $\mathcal{VT}^k$.

Atomic Broadcast in *TTP/C*. Special pieces of hardware (the *guardian*s) prevent nodes from sending outside the assigned slots and ensure that all receiving nodes have a consistent view of what has been sent. *TTP/C* autonomously and deterministically exchanges state messages between Communication Network Interfaces (CNIs) in the nodes of a distributed systems. State message semantics [Kop97, p.33] means that only state values are transmitted and the communication system itself performs control on when to exchange messages. In contrast, event messages are messages that only contain the difference between two states. When event messages are transmitted, the order of these messages can be recovered by timestamping messages and ensuring that event messages are not overwritten in the CNI.

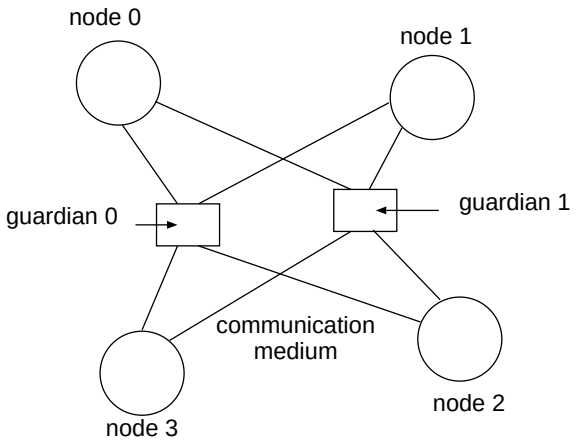


Figure 2.10: TTP/C cluster (star architecture) with four nodes and two guardians

The guardian design ensures the consistency of message data. The basic idea of the guardian design – which is described in detail in [BS01, BKS02] – is that replicated guardians decide about the correctness of a node and other nodes accept this decision.

In detail: in the time domain, a guardian checks that communication of other nodes starts and ends in the intervals starting one precision before the assumed sending time and ending one precision afterwards, respectively. All nodes accept any incoming traffic in the interval twice the amount of the precision around the assumed sending time, because the guardians and the sending

node can only be synchronized with Π. If a node does not obey to this rule, its message is blocked and/or marked as incorrect. The guardians decide centrally about the correctness of a node and other nodes accept this decision in order to avoid inconsistencies among nodes.

In the value domain, a guardian performs signal reshaping in order to avoid any inconsistencies between nodes. The principle for the value domain is similar to the time domain: the tolerances of the guardians are much tighter than the tolerances of the nodes. Consequently, when a guardian accepts a value of a node on the communication medium all nodes will accept it.

Due to the design of the nodes, the replicated guardians, the fault-tolerant internal clock synchronization, and the static access pattern, all correct nodes will consistently receive all messages in the same order. Figure 2.10 depicts a cluster with guardians. For a detailed discussion, please refer to [BKP01] for the design and fault hypothesis of guardians and to [SP02, SB02] for startup of the system.

Common-Mode-Drift Correction in *TTP/C*. *TTP/C* provides a mechanism to correct the common-mode drift of the global cluster time. For the correction, a correction term is added to the correction value of the internal clock synchronization, which is calculated by the FTA algorithm, every TDMA round at each node. This resulting value is used for the correction of the virtual clock times for the next TDMA round. This mechanism, which adjusts the virtual clock time at the same clock tick instances at every node, can be used as basic correction primitive for the external clock synchronization of a cluster and for inter-cluster clock synchronization.

2.4 Summary

This chapter gave an overview of the concepts and terms that will be used throughout this thesis. We started off by defining distributed systems in general. Further, we introduced the notions of the impairments, the means, and the attributes of dependability and put particular emphasis on how fault tolerance can be achieved. Concerning fault tolerance, we described the two main approaches to reach fault tolerance: replication of components and self-stabilization. This chapter then introduced some aspects of distributed systems, such as synchrony, composability, and types of broadcast message deliveries. In the context of distributed systems, real-time, multi-cluster, and embedded systems have been described.

The second major section described the concept of clocks and the principle of clock synchronization and defined measures, such as drift rate, precision,

accuracy. It also classified clock synchronization algorithms and introduced time standards and sources. Time standards are agreed notions of time. Time sources can be used to access these time standards. We described aspects of the use of time by applications, such as time representation, global, reasonable, and sparse time. The chapter ended with an brief explanation of a fault-tolerant embedded system architecture, the Time-Triggered Architecture with its communication protocol *TTP/C*.

Chapter 3

Problem Statement and Objectives

The real problem is what to do with the problem-solvers after the problems are solved.

Gay Talese

Chapter 2 introduced the reader to the concepts and terms used in this thesis, namely that of clocks, clock synchronization, distributed, embedded, and multi-cluster systems. Starting from a precise notion, this chapter concisely states the problem this thesis addresses. We also justify the research work by discussing what known clock synchronization approaches have not yet dealt with and what are the needs of a clock synchronization algorithm for embedded systems.

3.1 Problem Statement

Typical large-scale distributed fault-tolerant real-time systems can be realized as multi-cluster systems. Multi-cluster systems consist of a number of groups of components with a high inner functional and temporal coupling, so-called clusters. The clusters are, in turn, interconnected. The coupling between clusters is, however, much looser than the coupling within the single clusters. To meet the temporal requirements of a fault-tolerant real-time (control) application the components of a cluster, which share the real-time workload, need a

common fault-tolerant time base. Such a time base is established by performing fault-tolerant internal clock synchronization.

The goal of this thesis is the clock synchronization of multi-cluster systems used as embedded control systems. Multi-cluster clock synchronization can be seen as external clock synchronization of individual clusters. While in the traditional model of external clock synchronization the drift rate of relational clocks is zero with respect to real time [CF95], the relational clocks in multi-cluster synchronization are allowed to have a maximum drift rate. This thesis shows that multi-cluster synchronization of internally synchronized clusters can be achieved by developing an external clock synchronization algorithm that allows relational clocks to have a bounded drift rate. This drift rate of relational clocks can be as large as the drift rate of the global time of a cluster. As the drift rate of relational clocks is regarded as non-zero, a number of additional questions arise:

- Clock drift rate errors are characterized by a stochastic and a systematic error [Sch88]. Clocks are the basis for the global time. Is the clock drift rate of the global time also characterized by a stochastic and a systematic part? If so, to what level can the systematic drift rate of the global time be compensated by a multi-cluster clock synchronization algorithm?
- What level of accuracy (as a quality measure) can be achieved by a multi-cluster clock synchronization algorithm?
- The global time of a cluster is represented by (not necessarily all) clocks of a cluster. How does a change of a clock set effect the drift rate of the global cluster time and, subsequently, the multi-cluster clock synchronization algorithm?
- The analysis of current clock synchronization algorithms takes one error term (clock reading error; that comprises all errors, such as reading of clock, jitter of communication system, digitalization errors) into account [FC97]. Does an analysis that differentiates between the different error types lead to additional insights? How do architecture properties influence the quality measures of a clock synchronization algorithm?

3.2 Objectives

The objective of this thesis is the development of a multi-cluster clock synchronization algorithm for embedded control systems. This algorithm must have the following properties in order to tackle the requirements of clock synchronization in embedded control systems:

Fault tolerance. A clock synchronization algorithm must be fault-tolerant in order to be able to cope with the demanding dependability requirements of embedded systems due to autonomy needs. From the described fault tolerance techniques in Section 2.1.1, fault masking and fault recovery are appropriate techniques for handling faults in a clock synchronization algorithm.

Fault masking. Faults described in the fault hypothesis must be masked for applications, because these can be frequently occurring faults leading to system failures.

Fault recovery. As faults cannot always be predicted or be masked with justifiable overhead, mechanism for tolerating rarely occurring faults must be supported. These mechanisms aim at stabilizing the system in a state from which correctly performing systems can start again. Such a mechanism is, for example, self-stabilization.

Tight synchronization. The quality of synchronization between clocks of nodes of the different clusters must be in the same order of magnitude as the quality of synchronization of clocks of nodes of the same cluster, in order to be able to collectively control the environment with high quality. The maximum difference between any two clocks of a system determines the quality of a system-wide image of the environment and, subsequently, the quality of control of the whole system, because it is a limiting factor of a sparse system-wide time base.

Non-interference. The precision is the maximum difference between any two clocks of nodes of one cluster and determines time-specific parameters of control applications. Multi-cluster synchronization influences the precision of a cluster as does external clock synchronization. This is inevitable due to the impossibility of perfect clock synchronization [LL84b], which leads to a different view of correction values of clock synchronization algorithms. Yet, the influence of an external or a multi-cluster clock synchronization algorithm on the precision of a cluster, first, should be minimized and, secondly, should be negligible compared to influences of other errors. Furthermore, the influence should only stem from node properties, and not others – especially not parameters outside the control of a node – for two reasons. First, failures in the temporal domain related to synchronization can be isolated. Secondly, composability of cluster time bases is enabled.

We call a clock synchronization algorithm where the influence of clock synchronization on the precision depends only on parameters of nodes a *non-interfering* clock synchronization algorithm, because synchronization does not interfere with properties building on the precision. Syn-

chronization algorithms where the influence on the precision of a cluster stems from components not belonging to the cluster, such as the maximum deviation of external clock readings, are called *interfering* clock synchronization algorithms. Non-interference ensures that applications can build upon properties of a time base that will not change by the composition and synchronization of several clusters.

Low computational complexity. Embedded control systems have only low computational power compared to other computer systems, such as desktop PCs, and stringent memory restrictions as elaborated in Section 2.1.3. Thus, an algorithm designed for deployment in embedded systems must be able to deal with the stringent resource conditions and have low computational complexity.

Work on clock synchronization has been discussed and classified in Section 2.2. None of the presented algorithms explicitly takes drifts of relational clocks into account. Related to multi-cluster clock synchronization is the work of Fetzer and Cristian [FC97, CF95]. They present a fault-tolerant external clock synchronization algorithm where the influence of external clock synchronization on the precision depends not only on node-specific parameters, such as the maximum deviation between external clock readings. This influence stems from the fact that nodes can have an inconsistent view of an external clock reading value. Fetzer and Cristian bound the influence of the external clock synchronization algorithm on the precision by restricting the maximum correction to the maximum drift of nodes and by assuming initial synchronization. Kopetz et alii [KHK+96, KKMS95] address the problem of how to integrate external and internal clock synchronization in the context of synchronizing multi-cluster real-time systems. They consider only one relational clock and, thus, cannot achieve fault-tolerant synchronization. Harper et alii [HLD88] address the problem of clock synchronization of clusters in parallel systems. As addressed in Chapter 2, the difference between parallel and distributed systems is the unreliable communication of distributed systems.

3.3 Summary

This chapter addressed the problem this thesis tackles by giving a precise problem specification and presenting a requirements analysis of clock synchronization algorithms for embedded control systems. Known clock synchronization algorithms that are related to the presented research work were discussed.

Chapter 4

Framework and Assumptions

Omnia, Lucili, aliena sunt, tempus tantum nostrum est.

Seneca, Epistulae morales 1,3

Any concept is a matter of abstraction. Its goal is to explicitly distinguish relevant from irrelevant facts and properties. This chapter describes the underlying concepts of the research work of this thesis. This is done by presenting the framework of multi-cluster clock synchronization and the underlying assumptions of the multi-cluster clock synchronization algorithm.

4.1 Framework

Multi-cluster synchronization is achieved by performing *external clock synchronization of global times of individual clusters* using relational clocks as references. As described in Section 2.2.1, relational clocks are not ideal representations of real time and, thus, have a bounded drift rate different from 0 and an accuracy with respect to real time that is different from 0. A relational clock is a virtual clock that is used as a reference. Relational clocks can either be synchronized to a time source or not[1]. A node that can read the time of a relational clock in addition to its virtual clock time is called *time master node.* A time master node can calculate the difference between its virtual clock time

[1] A time source is also called *external time source*, since – from a cluster-point-of-view – a time source provides time information to a cluster that is outside the control of a cluster.

and the relational clock time. This difference is also called offset and is denoted Ω_t^j for time master node j at time t. Figure 4.1 depicts a multi-cluster system with three clusters.

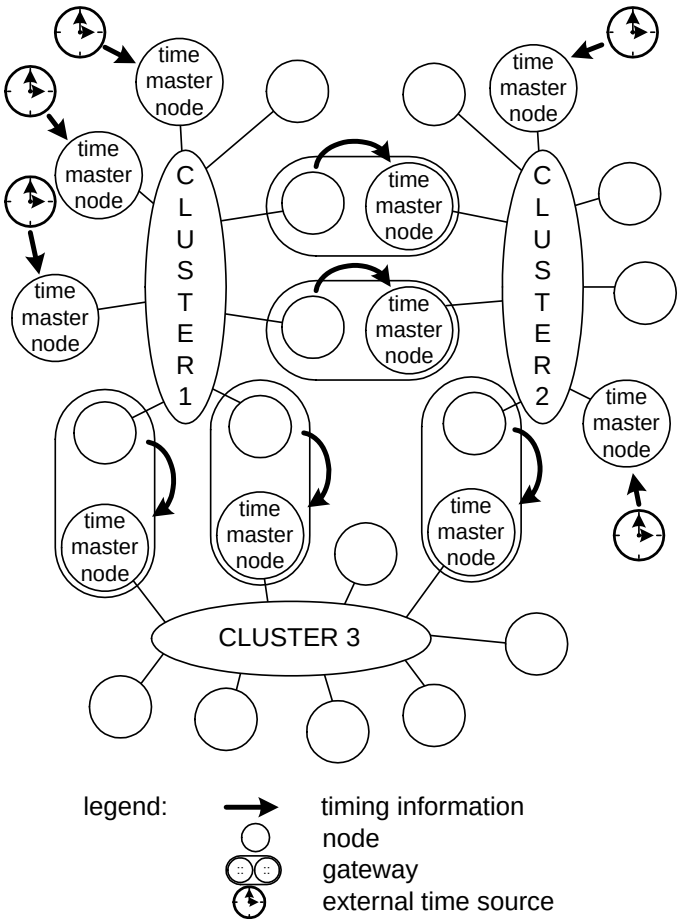


Figure 4.1: Multi-cluster system

External and Inter-Cluster Synchronization. In this thesis, we differentiate between synchronization to relational clocks that are synchronized to a time standard and to relational clocks that are not synchronized to a time standard, because if relational clocks are synchronized to a time standard conclusions about the synchronization quality to real time can be drawn. If a relational clock that is synchronized to an external time source is used as relational clock, we call this *external synchronization*. If a relational clock is not synchronized to a time standard, we call this *inter-cluster synchronization*. For inter-cluster synchronization, virtual clocks of other clusters are used as references. Since virtual clocks build global cluster times (see Section 2.2.4), we say

for short that inter-cluster synchronization is concerned with synchronizing a global cluster time to other global cluster times.

For inter-cluster synchronization, timing information is conveyed from one cluster to another cluster via dedicated components, called *gateways*. A gateway consists of two nodes. One node is a participant of one cluster with a virtual clock time that is internally synchronized to the other virtual clock times of this cluster. The other node is participant of the other cluster and also has a virtual clock time, which is internally synchronized to the virtual clock times of the other cluster. The virtual clock time of each node is a representation of a global cluster time (see Section 2.2.4). For external synchronization, relational clocks synchronized to time standards provide timing information to nodes of a cluster. These external time sources (also called *time servers* [KKMS95]) consist of a virtual clock time that is synchronized to a time standard, such as GPS time.

If synchronization to an external time source is necessary, we assume that all relational clock times are in relation to real time as indicated in Figure 4.1, because relational clock times are directly or indirectly synchronized with an external time source that is synchronized with real time. Direct access to external time sources enables the smallest achievable accuracy with respect to real time. The relational clock times of cluster 1 in Figure 4.1 have direct access to external time sources. For clusters where the relational clock times are not directly externally synchronized, the accuracy with respect to real time increases by the precision of the clusters in-between the relational clock times and the external time sources. For example, the smallest accuracy that can be achieved by a relational clock time of cluster 3, is the sum of the accuracy of the relational clock times of cluster 1 or 2 and the precision of cluster 1 or cluster 2, respectively.

Synchronization in Multi-Cluster Systems

The following sections focus on a description of fault-tolerant external clock synchronization of one cluster using relational clock times as timing information. Fault tolerance is achieved by replication of relational clock times, which are accessible via time master nodes. As described above, multi-cluster clock synchronization is achieved by synchronizing the global time of a cluster to relational clock times.

4.1.1 Flow of Timing Information

The following requirement at a multi-cluster level enables synchronization of clocks in multi-cluster systems when the algorithm described in Chapter 5 is

used:

Requirement 1. (Avoidance of Circular Dependencies between Cluster Times in Clock Synchronization) *In a multi-cluster system with M clusters ($M \geq 1$), if global time VT^{m}, $1 \leq m \leq M$, synchronizes using relational clock time $\mathcal{RCT}^j$, $\mathcal{RCT}^j$ must not use VT^{m} directly or indirectly as synchronization reference. In other words: when synchronizing, circular mutual dependencies of cluster times (as indicated in Figure 4.2) are avoided.*

Requirement 1 ensures the stability of cluster times, because it avoids feedback loops and associated control problems. Referring to Figure 4.2, the global cluster time of the upper-most cluster(s) is first synchronized, then the next level achieves the targeted accuracy value and so on, because the global cluster times synchronize towards their relational clock times.

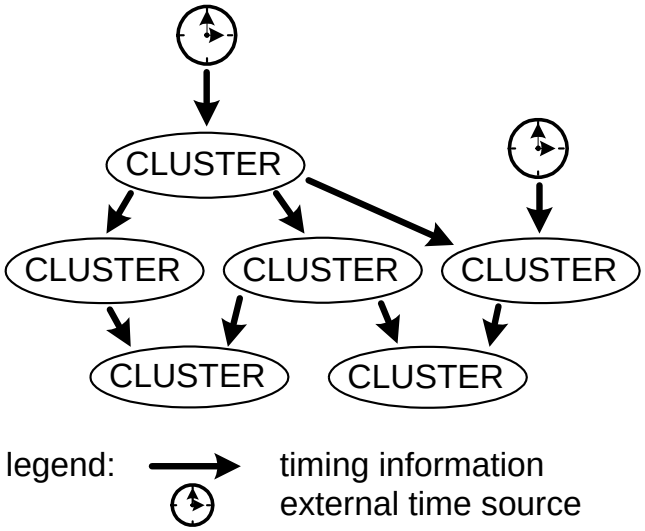


Figure 4.2: Propagation of timing info in a multi-cluster system

All relational clocks provide the timing information in a format as described in Section 2.2.4. This uniform representation of time in a multi-cluster system avoids misinterpretations. If the synchronization of the multi-cluster system to real time is not required, no external time sources have to be used as relational clocks. The time of the first cluster in the synchronization network is then the reference time for all other clusters (this is the uppermost cluster in Figure 4.2). This is a restriction of the flow of timing information, which leads to a solution of multi-cluster synchronization that does not have to address discrete control problems at multi-cluster level. A PhD thesis that addresses especially the problem of conveying timing information in a cyclic manner is currently under development at our institute (Institut für Technische Informatik, Technische Universität Wien, Vienna Austria).

In the latter of this thesis, we focus on the evaluation of clock synchronization either of two adjacent cluster (directly connected by gateway nodes) or of clock synchronization of one cluster to an external time source. Clock synchronization of multiple clusters is achieved by synchronizing several adjacent clusters and/or by synchronizing to external time sources as described in the next section.

4.2 Architecture of One Cluster

A cluster consists of N nodes. Nodes are considered to be fault containment regions. M of these nodes serve as *time master nodes*. Without restriction of generality, time master nodes are numbered from 1 to M and other nodes from $M+1$ to N. Time master node j has access to the relational clock time $\mathcal{RCT}^j$, $1 \leq j \leq M$. A time master node j measures the difference between its virtual clock time $\mathcal{VT}^j$ (depicted as small digital clock on the lower end of the circle in Figure 4.3) and its relational clock time $\mathcal{RCT}^j$ (large digital clocks in Figure 4.3) at time t. This difference is called *offset*, denoted Ω_t^j. This offset of a time master node is sent to all nodes via the communication system as shown in Figure 4.3 and described in detail in Section 2.2.2.

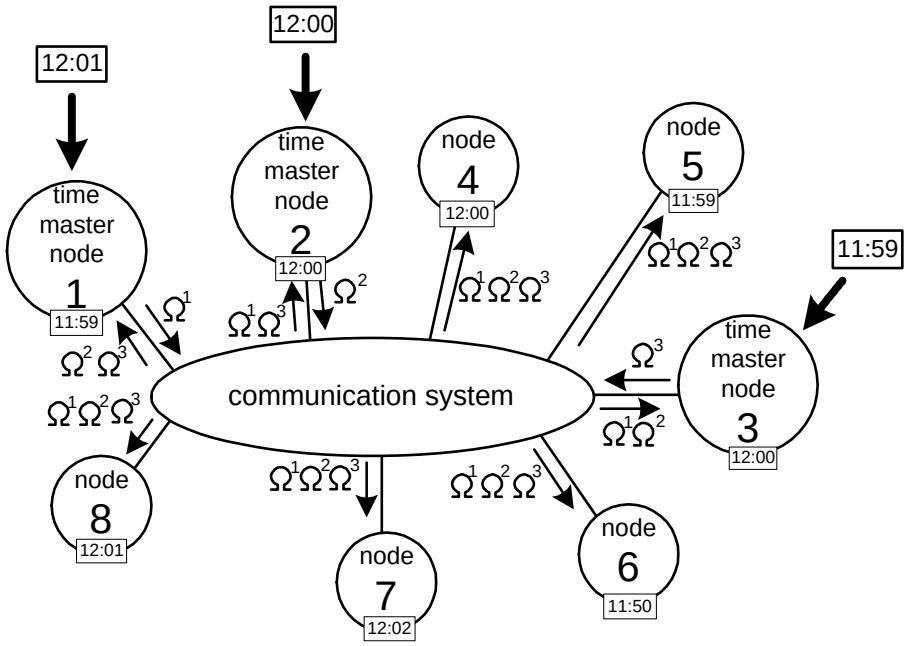


Figure 4.3: Cluster with eight nodes, three of the nodes are time master nodes ($N = 8, M = 3$)

4.3 Requirements

For the fault-tolerant clock synchronization, it is assumed that the following requirements are satisfied for each cluster.

4.3.1 Communication System Requirements

Requirement 2 (Atomic Broadcast). *The communication system provides atomic broadcast [HT94] despite of F arbitrary faults of links or nodes. Link failures are mapped as node failures.*

Requirement 3 (Λ-Timeliness). *There is a known constant Λ such that if a message is broadcast at t all correct nodes will receive this message at latest at $t + \Lambda$ [HT94].*

Since Requirement 2 asserts that the communication system handles up to F arbitrary failures without loss of service, a faulty node will not be able to keep correct nodes from sending as long as there are no more than F failures. Requirement 3 assures that all correct nodes receive the same set of messages within a bounded duration. As each time master node sends the difference between its virtual clock time and relational clock time (its offset) in a broadcast message, all correct nodes of a cluster have the same set of offset values.

4.3.2 Synchronization-Related Requirements

Requirement 4 (Relational Clock Times). *All M relational clock times of a cluster, $\mathcal{RCT}^i$, $1 \leq i \leq M$, are synchronized within Δ.*

Please note, if a cluster uses relational clock times that are virtual clock times of different clusters, these relational clock times can be "far apart" at the beginning of external clock synchronization, because these relational clock times are not initially synchronized. Still, Requirement 4 is satisfied and the difference between relational clock times is bounded, although the value of Δ may be large.

Requirement 5 (Fault-Tolerant Internal Clock Synchronization). *The internal clock synchronization algorithm tolerates up to F arbitrary faults of nodes or links.*

Requirement 5 ensures that a faulty node does not effect properties of the internal clock synchronization, such as the precision of a cluster, as long as there are at most F faulty nodes[2].

Requirement 6 (Common Notion of Time). *All correct nodes of a cluster have a common notion of time.*

A common notion of time means that the virtual clock times of nodes of a cluster are internally synchronized to each other with precision Π (as long as Requirement 5 is satisfied).

4.3.3 Tolerance to Faulty Relational Clock Times

A time master node is considered to be faulty, if either it performs incorrect operations or its relational clock time is faulty, that is not synchronized with other relational clock times. Faulty relational clock times are tolerated by replication of time master nodes. As one time master node has access to exactly one (distinct) relational clock time, replication of time master nodes implies replication of relational clock times.

Requirement 7. (Number of Time Master Nodes) *For correct external synchronization at least $2 \cdot F + 1$ time master nodes are present to tolerate F arbitrarily faulty time master nodes.*

$2 \cdot F + 1$ (and *not* $3 \cdot F + 1$) is the minimum number of time master nodes needed to tolerate F arbitrarily faulty time master nodes, because a node can take advantage of the information that all correct external relational clock times are synchronized within an interval of duration Δ (Requirement 4).

The following considerations prove that Requirement 7 assures that with $2 \cdot F + 1$ time master nodes and at most F faulty time master nodes or communication system failures, the median of all offset values is a correct offset value or one that can be considered as correct. A detailed proof why $2 \cdot F + 1$ is the minimum requirement for fault-tolerant external clock synchronization can be found in [FC97].

Relational clock times are at most Δ apart, because relational clock times are either hierarchically dependent on each other (as described in Section 4.1.1) or synchronized to a time standard. Furthermore, the virtual clock times of time master nodes of one cluster are synchronized within Π. Consequently, all

[2]Please note that the precision of a cluster is the maximum deviation between any two correct nodes. As these deviations can be influenced and the values of the deviations increased by faulty nodes, the precision – as a maximum value – comprises the influence of faulty nodes.

correct offset values differ by at most $\Delta + \Pi$. If there are at least $2 \cdot F + 1$ time master nodes, there are at most F faulty offset values and at least $F + 1$ correct offset values. If all offset values are sorted by value, the median of the offset values is either a correct value or one that "lies" between correct values, which can be considered as correct. As correct values are the interval $\Delta + \Pi$, the median is also within this interval. Consequently, $2 \cdot F + 1$ time master nodes suffice to choose a "correct" offset value, which can be used for synchronization.

4.3.4 Requirements Regarding the Drift Rate and Correction of the Global Time of a Cluster

Requirement 8. (Systematic Part of Global Cluster Time Drift Rate) *The systematic part of the drift rate of the global cluster time, denoted ρ_{sys}^{VT}, is approximately* constant *over a given period of time.*

Requirement 9. (Stochastic Part of Global Cluster Time Drift Rate) *The stochastic part of the drift rate of the global cluster time, denoted $\rho_{stoch}^{\mathrm{VT}}$, approximately follows a symmetric distribution function in a given period of time.*

Requirements 8 and 9 enable the estimation of the systematic part of the drift rate of the global time of a cluster in a given period of time. This can be accomplished by measuring the drift offset of the global cluster time at points in time (with equidistant intervals in-between) and calculating the drift rate by using the average of the measurements in the observation period. In the observation period, a representative number of measurements is collected.

Requirement 10 (Correction of Global Cluster Time). *All nodes of a cluster must be able to correct their virtual clock times with a correction rate that is larger than the sum of the maximum drift rate of the global cluster time and the maximum drift rate of the relational clock times.*

Requirement 10 enables the synchronization of the global cluster time even if the global cluster time and the relational clock times drift maximally and the global cluster time is not synchronized initially. This requirement is fundamental for being able to initially start nodes non-synchronized. Interestingly, some clock synchronization algorithms assume initial synchronization and bound the maximal possible correction to the maximum drift rate, such as [FC97, CF95]. This assumption alleviates analysis of the algorithm.

4.4 Summary

This chapter briefly explained the framework and the assumptions of multi-cluster clock synchronization. Synchronization of multiple clusters is achieved by individually externally synchronizing a cluster using the global cluster time of other clusters or external time sources as reference. We require that circular mutual dependencies between global times of clusters are avoided. Since we aim at deployment of the algorithm in fault-tolerant real-time systems, time and broadcast constraints concerning the communications systems have been made and that a cluster has already established a common notion of time. In order to be able to differentiate between stochastic and systematic errors of clocks, the systematic error must be constant and the stochastic one must follow a symmetric distribution. At last, assumptions about the possible correction rate were made.

Chapter 5

Multi-Cluster Clock Synchronization

A person with one watch knows what time it is;
a person with two watches is never sure.

PROVERB

In Chapter 4, we have presented the framework for the multi-cluster clock synchronization. That is, multi-cluster clock synchronization is performed by an external clock synchronization of one cluster that can use relational clocks as reference. In this chapter, we present the principle of operation of this external clock synchronization algorithm and analyze it.

The developed external clock synchronization algorithm is round-based and addresses the systematic and the stochastic part of drift rates differently. While the systematic part of the drift rate of the global time of a cluster, denoted ρ_{sys}^{VT}, is measured over a longer period and then corrected, the stochastic part, denoted ρ_{stoch}^{VT}, is immediately corrected.

The basic idea of the presented external clock synchronization algorithm is the following: Since all nodes of a cluster have a common notion of time due to internal clock synchronization (including time master nodes), all time master nodes measure the deviations between their virtual clock times and their relational clock times (these differences are called *offsets*) at predefined points in time. Due to the common notion of time nodes perform the measurements at about the same time. Nodes send these offsets to all nodes. Since (due to

atomic broadcast) all correct nodes receive the same set of offset values, they deterministically choose the same correction value. Due to the common notion of time, nodes correct their virtual clock times with the same amount at predefined points in time, again at about the same point in time. This ensures that the influence on the precision of the cluster will not depend on parameters other than the nodes' parameters. The following paragraphs describe the algorithm in more detail and analyze it.

5.1 Principle of Operation

The external clock synchronization algorithm is round-based, each round is executed in four phases; all nodes are in the same phase at about the same time. After finishing the last phase, all nodes begin again with the first phase.

Measurement of Offset Values

In the first phase, all M time master nodes measure the differences between their virtual clock times and their relational clock times at a predefined point in time, denoted $t_{def,n}$ where n is the number of the current execution round. Formally, at $t_{def,n}$, time master node j measures the offset $\Omega^j_{t_{def,n}}$, which equals $\mathcal{VT}^j(t_{def,n}) - \mathcal{RCT}^j(t_{def,n})$. The duration between succeeding measurement points ($t_{def,n}$ and $t_{def,n+1}$) is called *measurement interval*, denoted $R_{measure}$, is equidistant, and equals $t_{def,n+1} - t_{def,n}$.

Time master nodes meet the following three properties: First, time master nodes use their virtual clock times to determine the predefined points in time; secondly, virtual clock times are internally synchronized within Π; and, thirdly, all relational clock times of a cluster deviate by at most Δ. Consequently, all correct offset values differ by at most $\Delta + \Pi$.

Sending of Offset Values

In the second phase (after the measurement), time master nodes send their offset values to all cluster nodes. Due to the atomic broadcast (Requirement 2), all correct nodes receive the values of time master nodes and obtain a consistent view of the offset values. That is, in execution round n all nodes receive at least $\Omega^1_{t_{def},n}$, $\Omega^2_{t_{def},n}$, ..., $\Omega^K_{t_{def},n}$, where K is the number of received offset values[1].

[1] All nodes receive at least $M - F$ values and at most M values depending on the actual number of faulty time master nodes that do not send offset values and the communication system failures; $M - F \leq K \leq M$, where M is the number of time master nodes and F the maximum number of faulty time master nodes.

Requirement 3 puts an upper bound on the broadcast time of the offset values.

Agreement and Calculation of Correction Value

In the third phase, all nodes sort all received offsets by value and choose the median as basis for the calculation of the correction value. As each correct node has the same set of offset values, each correct node calculates the same median value. In measurement interval $n+1$, each node uses the calculated correction value of measurement interval n to correct its virtual clock time.

The calculated correction value in measurement interval n (which is used for correction in measurement interval $n+1$) is the sum of the median value and the estimated amount of the drift offset due to the systematic part of the drift rate[2] of the global time in measurement interval n. The estimate of ρ_{sys}^{VT} is initially zero. Every H^{th} measurement interval, each node calculates the average of the last H median values. The sum of the current estimate of ρ_{sys}^{VT} and the calculated average reflect the estimate of ρ_{sys}^{VT} used for calculation of the correction in the next H measurement intervals. We call the interval between two consecutive calculations of an estimate of ρ_{sys}^{VT} the *external clock synchronization interval*, denoted $R_{ext.CS}$. H is the called the *history* of the algorithm ($H = \frac{R_{ext.CS}}{R_{measure}}$).

The above described algorithm is refined in order to assure a bounded drift rate of the global cluster time. To achieve a bounded drift rate, nodes bound the correction value. As a consequence, ρ_{sys}^{VT} is only estimated and corrected if the correction value is below a threshold value. For a proper operation of the estimation of ρ_{sys}^{VT}, the threshold value must be at least as large as the largest possible drift offset in a measurement interval. The upper bound of the threshold value determines the maximum drift and correction rate of the global time and is discussed in Section 5.2.1

Figure 5.1 depicts the computation of the correction value (*nCorrTerm*) in each measurement interval. *nSysDriftOffset* denotes the estimate of the amount of the drift offset due to the systematic part of the drift rate of the global cluster time in a measurement interval.

[2]The difference between the drift rate and the drift offset of a clock is subtle. The drift rate equals the drift offset, if the period in which the offset is measured equals a normed interval, which is usually one second. In this thesis, we prefer the term drift rate when we compare two or more clocks. However, it is clear that one can only directly measure the drift offset of a clock time in an interval and calculate the drift rate of this clock afterwards. If the measurement interval equals the normed interval, the computation of the drift rate can be skipped and the value of the drift offset equals the value of the drift rate.

```
proc Initialize
    empty history buffer
    nSysDriftOffset = 0
.
proc upon reception of new offset values
    calculate median of offset values
    if value smaller than specified bound then
      add value to history buffer
    fi
    if history buffer full then
      compute average of history buffer values
      empty history buffer
      nSysDriftOffset += average
    fi
    nCorrTerm = median of offset values + nSysDriftOffset
    bound nCorrTerm
.
```

Figure 5.1: Pseudo code describing the computation of the correction value

Correction

In the fourth phase, all correct nodes use rate correction (see Section 2.2.2) for correcting their virtual clock times. A node performs rate correction by correcting the amount C at predefined points in time. C is the correction term per correction interval and is calculated as shown in Equation 5.1 for offset values measured at $t_{def,n}$.

$$C = \left(median_{1 \leq j \leq K} \left(\Omega^{j}_{t_{def,n}} \right) + \tilde{\rho}^{VT,m}_{sys} \right) \cdot \frac{R_{corr}}{R_{measure}} \tag{5.1}$$

where R_{corr} denotes the interval between succeeding corrections, K the number of received offset values[3], and $\tilde{\rho}^{VT,m}_{sys}$ the estimate of the systematic part of the drift rate of the global time of the cluster in measurement interval n, which is part of the external clock synchronization interval m. The function *median()* calculates the median of the offset values.

Nodes use their virtual clock times to determine the predefined instants at which the virtual cluster time is corrected. R_{corr} is much smaller than $R_{measure}$.

[3]Please note that K is the same for all nodes due to Requirement 2.

Startup and Reintegration

The following paragraphs describe an extension to the algorithm that allows a node to integrate and startup without initialization of data structures. This extension also enables the self-stabilization of the algorithm, because data structures of self-stabilizing algorithms do not have to be initialized [Sch93a].

A time master node sends an offset value instead of its relational clock time (which could then be used at each node for calculation of the difference between the relational clock time and the virtual clock time of a node) for three reasons: first, the amount of data needed to send the offset, which is the difference between two time values, is smaller than a time value, because the integer value that represents the difference is usually much smaller than the integer value that represents the time of a relational clock. Secondly, the offset value of a time master node is consistent at all correct nodes, while the difference between a relational clock and the virtual clock of a node is not consistent due to the impossibility of perfect synchronization of virtual clock times. The offset value is consistent at all correct nodes due to Requirement 2 and the availability of a common notion of time at all correct nodes. Thirdly, performing the calculation of the difference only at time master nodes is – from a cluster's point-of-view – less computation-intensive compared to a solution where each node calculates the differences between its virtual clock time and the relational clock times.

In order to assure that recovering or starting up nodes can integrate and get a value of the current estimate of the systematic part of the drift rate of the global time of a cluster, each time master node periodically broadcasts its current estimate of the systematic part of the drift rate of the global time (in addition to the offset value) in the second phase (Phase Sending and Agreement). In addition, each node performs a majority vote on the received current estimates of the systematic part of the drift rate of the global time of time master nodes and updates the local current estimate of the systematic part of the drift rate of the global time with the outcome of the majority vote. Voting and updating is only performed when at least $F+1$ time master nodes have sent an estimate of the systematic part, where F is the maximum number of faulty time master nodes. This avoids that faulty time master nodes can disturb external clock synchronization, because at least $F+1$ ($= 2 \cdot F + 1 - F = F + 1$) of the $2 \cdot F + 1$ received values will be correct. As the correct values are a majority, the majority vote will provide a correct estimate of the systematic drift rate of the global time.

We call the interval between two consecutive broadcasts of the estimates of the systematic part of the drift rate of the global time of a cluster the *integration interval*. Integration intervals are integer multiples of external clock

synchronization intervals and nodes a priori agree, when the integration interval starts, that is when time master nodes must send the current estimate of the systematic part of the drift rate of the global time. Examples of agreed instants for sending are every full minute or every full second. Integration and external clock synchronization intervals are phase-synchronous, that is when the integration interval starts, the external clock synchronization interval must start.

As described above, time master nodes empty the history buffer, when it is full. As the external clock synchronization and the integration interval are phase-synchronous, correct nodes (in synchronous operation) empty their history buffers at the beginning of an external clock synchronization interval. If a node encounters that its buffer is not empty at the beginning of an integration interval, it empties its history buffer. This ensures that (correct) nodes start an integration interval at approximately the same time and that the history buffers of different (correct) nodes contain the same number of elements and can be consistently filled up again.

Emptying all buffers and updating the estimate of the systematic part of the drift rate of the global time ensures that – after at most one integration interval – the computation of the estimate is done at all nodes at approximately the same time and uses the same offset values.

Figure 5.2 depicts a pseudo code describing the time master operations and Figure 5.3 describes the integration code.

```
proc upon reaching the defined time for an offset calculation
    if time master node then
      offset = virtual clock time - relational clock time
      send offset
      if begin of integration interval then
        send estimate of systematic part of drift rate of global time
      fi
    fi
.
```

Figure 5.2: Pseudo code describing the time master operations

5.2 Analysis of Synchronization

In this section the external clock synchronization algorithm is analyzed with respect to its correction rate, the compensation of the systematic part of the drift rate of the global time, non-interference, and self-stabilization.

```
proc upon reception of new estimates of the systematic part of drift rate
    if number of received estimates >= (F+1) then
      perform majority vote on estimates
      if no majority then
        set estimate (nSysDriftOffset) to 0
      else
        set estimate (nSysDriftOffset) to majority vote result
      fi
    fi
.
```

Figure 5.3: Pseudo code describing the integration code; F denotes the maximum number of faulty time master nodes)

5.2.1 Maximum Drift and Correction Rates

This section discusses the maximum drift and correction rates of the global time of a cluster. Since real-time control applications rely and depend on a time base with a bounded drift rate, it is reasonable to limit the correction term per correction interval, denoted C, to a maximum, denoted $\bar{C}$. The maximum correction rate, denoted $\bar{\rho}^{corr}$, is the rate at which the external clock synchronization algorithm can maximally correct the global cluster time. Equation 5.2 depicts the relation between $\bar{C}$ and $\bar{\rho}^{corr}$.

$$\bar{\rho}^{corr} = \left| \frac{\bar{C} + R_{corr}}{R_{corr}} - 1 \right| \tag{5.2}$$

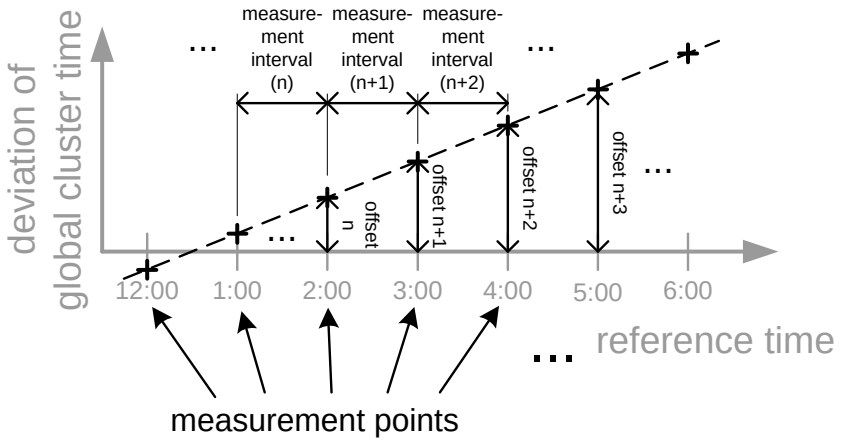
Let $\bar{\rho}^{VT}$ denote the maximum drift of the global cluster time not considering external clock synchronization and assume that there is no initial external synchronization of the cluster. In order to be able to compensate for $\bar{\rho}^{VT}$ and to synchronize the global cluster time, $\bar{\rho}^{corr}$ must be larger than $\bar{\rho}^{VT}$. $\bar{\rho}^{corr}$ determines the threshold value of the described algorithm.

If external clock synchronization is running, the global cluster time drifts at most $\bar{\rho}^{VT} + \bar{\rho}^{corr}$. There is a tradeoff between the time it takes to synchronize a cluster (if it is not initially synchronized) and the maximum drift rate $\bar{\rho}^{VT} + \bar{\rho}^{corr}$.

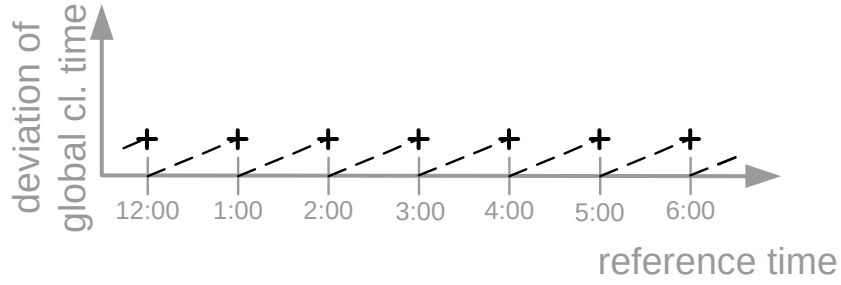
5.2.2 Compensation of Systematic Part of the Drift Rate

This section argues that all virtual clock times of a cluster are synchronized towards the relational clock times using the median of the offset values while

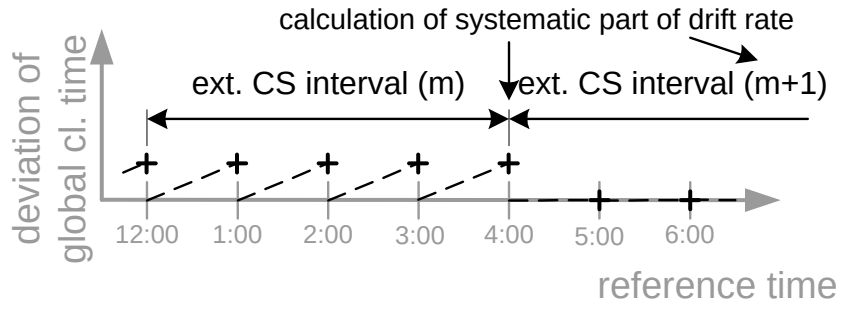
eliminating the influence of the systematic part of the drift rate of the global cluster time (denoted ρ_{sys}^{VT}).



(a) No correction



(b) Correction of median of offset values (agreed offset value)



(c) Correction of median and estimate of systematic part of drift rate

Figure 5.4: Deviation of global time of a cluster from a relational clock time using different correction techniques

Figure 5.4(a) depicts measurements of offsets if no correction is performed. If each node corrects an amount equal to the *agreed offset value* (which is the median value of all offset values) measured at the predefined time in execution round n (denoted $\Omega_{t_{def,n}}$) shortly after the measurement, the next measured and agreed offset value ($\Omega_{t_{def,n+1}}$) indicates the global cluster time drift ρ^{VT} during $R^{n+1}_{measure}$. Figure 5.4(b) depicts the deviation of the global cluster time from a relational clock time, if nodes correct just the agreed offset values.

If each node corrects the agreed offset value in the next measurement interval, the mean value of a sufficient number of agreed offset values ($\Omega_{t_{def,n}}$, $\Omega_{t_{def,n+1}}$, ...) will reflect a good estimate of the systematic part of the drift rate of the global cluster time (ρ^{VT}_{sys}) due to Requirements 8 and 9. $\tilde{\rho}^{VT,m}_{sys}$ denotes the estimate of ρ^{VT}_{sys} in $R^{m}_{ext.CS}$. If all nodes correct this estimate in addition to $\Omega_{t_{def,n}}$, $\Omega^{m}_{t_{def,n+1}}$ the next measurement will be close to zero. Figure 5.4(c) shows the deviation of global cluster time from a relational clock time where the history is 4 ($H = 4$); nodes also correct $\tilde{\rho}^{VT,m+1}_{sys}$ in the measurement intervals of the external clock synchronization interval $(m + 1)$ ($R^{m+1}_{ext.CS}$).

If ρ^{VT}_{sys} in $R^{m}_{ext.CS}$ deviates from $\tilde{\rho}^{VT,m}_{sys}$, then the mean of the H measured and agreed offset values will differ from zero and $\tilde{\rho}^{VT,m+1}_{sys}$ has to be updated to include the mean of the offset values as depicted in Equation 5.3.

$$\tilde{\rho}^{VT,m+1}_{sys} = \frac{\sum_{n=1}^{H} \Omega_{t_{def,n}}}{H} + \tilde{\rho}^{VT,m}_{sys} \tag{5.3}$$

This update is necessary after the external clock synchronization algorithm parameters are initialized after starting the algorithm or when the value of systematic part of the drift rate of the global time changes.

In Figure 5.4, state correction is used instead of rate correction to alleviate the understanding of the algorithm (this does not effect the behavior of the algorithm). Furthermore, the stochastic part of the drift rate of the global time (ρ^{VT}_{stoch}) is orders of magnitude smaller than systematic part (ρ^{VT}_{sys}). For the correction, rate correction should be used in order to ensure that applications can rely on a time without discontinuities.

Requirement 10 ensures that the algorithm can compensate the maximum drift of global cluster time and relational clock times and synchronize the global cluster time if it is not synchronized. Besides the fact that the Requirements 1 to 9 must hold, the threshold value for an agreed offset value to be used for estimation of the systematic part of the drift rate of the global time (ρ^{VT}_{sys}) must be greater than

$$\left(\bar{\rho}^{VT} + \max_{1 \leq i \leq M} \left(\bar{\rho}^{\mathcal{RCT}^i}\right)\right) \cdot R^{n}_{measure}$$

This allows the correct estimation of ρ^{VT}_{sys}.

In order to assure that interferences between internal clock synchronization and external clock synchronization are minimized, the same corrections of virtual clock times must be performed at the same ticks of virtual clock times. This is discussed in more detail in the next section.

5.3 Non-Interference

In this section, we analyze the non-interference of the external clock synchronization algorithm for F arbitrarily faulty nodes or communication links assuming the requirements of Section 4.3 are satisfied. Non-interference claims that the precision of the internal clock synchronization is not influenced by parameters that are not in the control of a node. We perform the analysis in two steps. At first, we prove that all correct nodes of a cluster consistently agree on *one* value for correction. Then, we show that the external clock synchronization influences the precision of a cluster only due to imperfect synchronization of virtual clock times, which results in a different view of the correction values at different nodes.

5.3.1 Consistent Agreement on One Correction Value

Property 1. *Only a faulty time master node does not send its offset value.*

Proof: Property 1 directly follows from Requirements 2 and 3. □

Please note, however, that a faulty time master node may send an incorrect offset value.

Property 2. *Correct nodes receive at least K offset values, where $M - F \leq K \leq M$.*

Proof: K denotes the number of offset values correct nodes receive. The proof of Property 2 is straight forward, since the fault hypothesis states that at most F time master nodes may be faulty. K equals $M - F$, if F faulty time master nodes do not send an offset. K equals M, if all time master nodes send values and due to the atomic broadcast (Requirement 2) all correct nodes consistently receive the same K $(= M)$ offset values. □

$M-K$ is the number of time master nodes that have not sent an offset value and are, thus, considered faulty. Please note that $M - K$ may be smaller than the actual number of faulty time master nodes, because up to $F - (M - K)$ nodes of the K time master nodes may have sent wrong offset values (it cannot be more than $F - (M - K)$ due to the fault hypothesis). A wrong offset

value cannot be detected by the communication system, thus, all correct nodes receive these values.

Property 3. *If there are at least* $2 \cdot F+1$ *time master nodes (i.e.,* $M \geq 2 \cdot F+1$*), the median is either a correct value (or calculated from two correct values, if* K *is even) or there exist at least one correct value that is larger and at least one correct value that is smaller than the median.*

Proof by contradiction: Say the median is a faulty value (or calculated using at least a faulty value) and the median does not lie between two correct values. Then, at least $\lceil \frac{K}{2} \rceil$ out of the K received offset values must be faulty. Since the $M - K$ time master nodes that have not sent a value are also known to be faulty, overall there are at least $\lceil \frac{K}{2} \rceil + M - K$ faulty time master nodes.

Let Equation 5.4 define the function of faulty time master nodes of the system with respect to K:

$$\mathcal{F}(K) = \left\lceil \frac{K}{2} \right\rceil + M - K \tag{5.4}$$

Property 2 gives the possible range of K, $(M - F) \leq K \leq M$. Since $\mathcal{F}(K)$ is monotonically decreasing, the minimum number of faulty time master nodes can be calculated by substituting K for M (Please note, this is the maximum influence of faulty time master nodes of the calculation on the median). This leads to a minimum number of faulty time master nodes: $\mathcal{F}(K) = \lceil \frac{M}{2} \rceil + M - M = \lceil \frac{M}{2} \rceil$. There is a contradiction between the minimum number of faulty time master nodes, which equals $\lceil \frac{M}{2} \rceil$, and Requirement 7 that allows at most $\lfloor \frac{M-1}{2} \rfloor$ time master nodes to be faulty. Consequently, Property 3 is valid. □

If there exists at least one correct offset value that is larger and at least one that is smaller than a faulty offset value, using this faulty offset value as reference for external synchronization will perform a successful external clock synchronization. From this observation, Property 3, and Requirement 7, we conclude the following: all nodes can take the median as a basis for the calculation of the correction term C and will externally synchronize. Since all nodes receive the same K offset values, all correct nodes compute the same median and take this median as basis for external clock synchronization.

5.3.2 Correction Does not Interfere with Precision

As described in Section 2.2.2, the internal clock synchronization uses the occurrence of virtual clock ticks as basis for correction.

For the external clock synchronization, all nodes correct their virtual clock times by the same amount at the same virtual clock tick due to the common

notion of time (Requirement 6). The correction amount is the same for all nodes (since all correct nodes choose the same offset value as shown in Section 5.3.1). The correction of the virtual clock times is performed by correcting the number of hardware ticks per virtual clock tick (as done in the internal clock synchronization). If a faulty node corrects a different amount than all correct nodes do, this can lead to a virtual clock time at this faulty node that is not internally synchronized anymore. Requirement 5 request that the internal clock synchronization algorithm must be able to tolerate up to F faulty nodes. Thus, if no more than F nodes are faulty, the external clock synchronization will not interfere with the precision of the internal clock synchronization and all correct nodes will externally and internally synchronize.

Please note: the different digitization errors that occur when digitizing the correction value at different nodes due to different hardware clock frequencies influence the amount of the actual correction. However, these errors do not influence the precision, because these hardware-dependent errors must already have been accounted for in the precision of the internal clock synchronization.

The only influence of the external clock synchronization on the precision is the error introduced due to the different views of the correction amount by different clocks, which is described in the next section.

5.3.3 Remaining Influence Due to Imperfect Synchronization

Distributed clocks cannot be perfectly synchronized [LL84b]. Different nodes will have a different view of the common correction value. E.g., say the correction term for all nodes is 5 μs, then for a clock that is 10 % faster than a perfect one, these 5 μs will be 4.5 μs ($= 5 \cdot \frac{100-10}{100}$ μs) measured by a perfect clock; for a clock that is 10 % slower than a perfect one, these 5 μs will actually equal 5.5 μs. The different views of the correction amount result in different corrections at different nodes and, thus, influences the precision of the cluster. This influence on the precision is proportional to the maximum drift between two virtual clocks, which may be as high as two times the maximum drift of hardware clocks ($2 \cdot \bar{\rho}^{\mathcal{HT}}$). Let $\bar{\epsilon}$ denote the maximum influence of the external clock synchronization on the precision. Equation 5.5 depicts the relation between $\bar{\epsilon}$ and $\bar{\rho}^{\mathcal{HT}}$.

$$\bar{\epsilon} = \bar{C} \cdot 2 \cdot \bar{\rho}^{\mathcal{HT}} \tag{5.5}$$

Since $\bar{\rho}^{\mathcal{HT}}$ is given, the minimum of $\bar{\epsilon}$ can be achieved by minimizing $\bar{C}$. If, in Equation 5.2, $\bar{\rho}^{corr}$ is held constant (since it is given by application requirements), minimization of $\bar{C}$ requires minimization of R_{corr}. The (theoretical)

lower bound for R_{corr} is the granularity of the VT, because the duration between two virtual clock ticks can be shortened or lengthened by hardware clock ticks due to the model of virtual clock time (see Section 2.2.1).

From this we conclude that in order to achieve a minimum influence of external clock synchronization on the precision of a cluster, an external clock synchronization algorithm must correct small correction amounts very often.

5.4 Self-Stabilization

As described in Section 2.1.1, self-stabilization is a way to provide fault-tolerant behavior of an algorithm in case of transient failures. These failures may have put the system into an illegitimate state and Requirements 2 to 10 may have been temporarily violated[4]. After a transient failure, we assume that Requirements 2 to 10 hold again.

In the described algorithm, illegitimate states are all states where the buffers of correct nodes are filled with different offset values, contain a different number of values, or where correct nodes have different estimates of the systematic part of the drift rate of the global time. A legitimate state is when all node buffers for the offset values contain the same number of offset values, node buffers contain the same offset values, and all nodes have the same estimate of the systematic part of the drift rate of the global time of a cluster. A legitimate state will eventually lead to a synchronized multi-cluster system, because all nodes equally correct the same amount in each measurement interval.

The following paragraphs argue that if nodes are in an illegitimate state after a transient failure, they will transit into a legitimate state after at least one integration interval and one external clock synchronization interval after Requirements 2 to 10 become valid again.

Property 4. *A correctly performing node that contributes to an illegitimate system state corrects its local state within at most one integration interval and one external clock synchronization interval and enables a system-wide legitimate state.*

Proof: A correctly performing node is a node that correctly executes the external clock synchronization algorithm, but may have a local state that is the

[4]This section does not analyze a violation of Requirement 1 for two reasons. First, this cannot happen temporarily, as the structure of the multi-cluster systems cannot be altered dynamically. Secondly, it is impossible to detect a circular dependance of cluster times, as the effects of a circular dependance are similar to the ones from one or more faulty time master nodes. As a consequence, self-stabilization is not an appropriate means for tolerating violations of Requirement 1.

"cause" of an illegitimate state. In order to prove Property 4, the three causes of an illegitimate state must be addressed. As described above, these three causes are: first, buffers contain a different number of offset values, secondly, buffers are filled with incorrect offset values; and, thirdly, different estimates of the systematic part of the drift rate of the global time exist at different nodes. Property 5, 6, and 7 argue that the described external clock synchronization algorithm enables a transition from an illegitimate system state to legitimate one. We will first prove these three properties before continuing the proof of Property 4

Property 5. *All correct nodes have equally filled buffers, that is, buffers have the same number of elements, within at most one integration interval.*

Proof: Buffers are emptied at the beginning of an external clock synchronization interval and each node adds one value in each execution round. As integration intervals and external clock synchronization intervals are phase-synchronous, it suffices to show that integration intervals of correct time master nodes start at approximately the same time at each node within one integration interval.

All nodes have a priori agreed on the start of an integration interval. All nodes check the actual beginning of the start of an integration interval and empty their buffers at the beginning. Consequently, all correct nodes have the same number of elements from the time on when node buffers have been emptied. □

Property 6. *All correct nodes have buffers with a consistent set of offset values within at most one external clock synchronization interval.*

Proof: Because all values in a node's buffer are overwritten by new median values (which must be correct due to Requirements 2 to 7) in one external clock synchronization interval and because an integration interval is larger than or equal to an external clock synchronization interval, all buffers contain correct values within one external clock synchronization interval after Requirements 2 to 10 hold again. □

Property 7. *All correct nodes have the same estimate of the systematic part of the drift rate of the global time within at most one integration interval.*

Proof: As argued in proof of Property 5, integration intervals of correct time master nodes start at approximately the same time at each node within one integration interval. Correct time master nodes send their estimates of the systematic part of the drift rate of the global time at the next integration interval start and nodes perform a majority vote on the estimates. Nodes

update their estimate of the systematic part of the drift rate of the global time, if there is a majority; otherwise, they set it to zero. Consequently, all correct nodes have the same estimate after one integration interval. □

Continuation of proof of Property 4:

As the update of the estimate of the systematic part of the drift rate of the global time with a correct estimate and the update of the buffer offset values occurs in parallel, it suffices to choose the longest one of the two tasks. The longest task determines the time when a legitimate state is reached. It takes at most one integration interval until all correct nodes have an update of the systematic part of the drift rate of the global time (Property 7). Yet, it takes at most one integration interval until all nodes have the same number of elements (Property 5). From the point in time when buffers have the same number of elements, it takes one additional external clock synchronization interval until all nodes have the same correct offset values in their buffers (Property 6). Consequently, it takes at most one integration interval and one external clock synchronization interval until all nodes have the same estimate of the systematic part of the drift rate of the global time, the same number of offset values, and the same (correct) offset values in their buffer and, thus, have reached a legitimate state. □

5.5 Resource Requirements

As the described algorithm is to be used in embedded control systems, which are restricted in resources (see Section 2.1.3), this section addresses the resource requirements of the computation-intensive parts of the described algorithm and compares the computational overhead and memory requirements with alternative solutions.

The calculation of the systematic part of the clock drift can be seen as finding a linear approximation of the drift rate using median values in an external clock synchronization interval. For this problem, linear regression analysis can be used. The memory requirements for the linear regression approach and the described algorithm are in the same order of magnitude. However, the running times differ largely for large history length. An evaluation of the growth of the running times as a function of the history length described using the O-notation [CLR90] yields the following results: using linear regression for the calculation of the systematic part of the drift rate of the global time is bound by $O(h^2)$ while the described external clock synchronization algorithm is bound by $O(h)$, where h is the history length. The difference in necessary computational overhead becomes important for large history lengths. The next section discusses the influence of different algorithm parameters.

The improvement of the external clock synchronization in computation time over the regression analysis comes with a slightly increased synchronization time, if clusters are not initially synchronized (one additional external clock synchronization interval) and the restriction that the interval between two offset measurements must be constant for the external clock synchronization while it can be variable when the regression analysis is used for the calculation of the systematic part of the drift rate.

5.6 Discussion of Algorithm Parameters

This section discusses the influence of parameters of the algorithm.

History length H. There is a trade-off between quick adaption after a change of the systematic part of the drift rate of the global time (ρ_{sys}^{VT}) and a precise estimate of ρ_{sys}^{VT}. A small H allows quick synchronization at startup and quick adaption after a change of ρ_{sys}^{VT}. A large value for H enables a precise estimate of ρ_{sys}^{VT}, because sufficient values represent the symmetric distribution of ρ^{VT}. H is always an integer number and should be a power of two to be able to implement the algorithm using (fast) shift operations instead of (slow) division operations.

Measurement interval $\mathbf{R_{measure}}$. A small $R_{measure}$ allows synchronization with a small accuracy due to frequent re-synchronizations. Frequent measurements and increasing computations, on the other side, increase the required computational power necessary for the external clock synchronization algorithm.

Integration interval. The integration interval determines the duration an integrating node has to wait until it receives an update of the current global time of a cluster. In the worst case, this duration can be as large as the integration interval. The longer the interval the longer is the waiting duration in the worst case. The integration interval also determines how much data is sent using the communication system for integration or startup purposes. The longer the interval the less data has to be sent. There is a trade-off between quick integration and small data amounts to be transferred.

5.7 Summary

This chapter describes the principle of operation. The multi-cluster clock synchronization algorithm operates as follows: each time master node measures

the difference of its virtual clock time and its relational clock time, sends it to all nodes. A node calculates a correction value. Due to the requirements, all nodes correct the same value. This chapter analyzed the drift and correction rates and the compensation of systematic clock errors. We showed that the algorithm is non-interfering, that is, the precision of a cluster is not influenced by external parameters. We also showed that remaining influence of the synchronization is negligible. Furthermore, the self-stabilization and resource requirements of the algorithm were analyzed. The chapter ended with a discussion on the algorithm parameters.

Chapter 6

Multi-Cluster Clock Synchronization in the Time-Triggered Architecture

It is impossible to mediate on time
and the mystery of the creative process of nature
without an overwhelming emotion at the limitations of human intelligence.

A.N. Whitehead

The previous chapter introduced the multi-cluster clock synchronization algorithm and argued that the algorithm is fault-tolerant by using replication and by its self-stabilizing design. Furthermore, the algorithm achieves tight synchronization by eliminating effects of the systematic part of the drift rate on the accuracy while keeping the necessary computational overhead low, that is the required computational power is increasing linearly with the history length of the algorithm. Non-interference of the algorithm enables composability and is achieved by providing a consistent set of offset values to all nodes of a cluster.

This chapter describes a case study, an implementation of the multi-cluster clock synchronization algorithm using the Time-Triggered Architecture (TTA) and its communication protocol TTP/C.

6.1 Goals

The goals of this case study are:

- The validation of the presented concepts by presenting an implementation of the multi-cluster clock synchronization algorithm on a system designed for embedded control applications using commercially available hardware.
- The verification that a constant systematic part of the drift rate of the global cluster time can be compensated by the external clock synchronization algorithm, if Requirements 8 and 9 hold.
- Examination of the external clock synchronization algorithm behavior if the systematic drift rate changes, as happens when the set of nodes that builds the global cluster time changes.
- Examination of the validity of Requirement 8 for the used architecture.
- Identification of parameters that influence the accuracy and investigation of changes of these parameters.

This chapter addresses these goals by first describing the implementation.

6.2 Implementation

As target hardware for the implementation, the Time-Triggered Architecture (TTA) has been chosen. *TTP/C* [Kop99], the time-triggered communications system of the TTA, provides atomic broadcast and internal clock synchronization services that meet Requirements 2, 3, 5, and 6 stated in Section 4.2.

The external clock synchronization algorithm has been implemented on two different *TTP/C* hardware boards – the TTPnode evaluation board and the TTPpowernode evaluation board. Both boards use the commercially available *TTP/C* controller in silicon, TTP/C C1 [Kop98], but different host processors – a Motorola MC68376 and Motorola PowerPC MPC555. A detailed description of the hardware details can be found in [TTT01].

Measurement of Offset Values

Both of the above mentioned Motorola processors are equipped with a Motorola Time Processor Unit (TPU). The TPU has been programmed to calculate the offset, that is the difference of a relational clock time and the virtual clock time

at a time master node. The granularity of the measurement unit ($g_{measure}$) – one TPU tick – is either a multiple of 190.7 ns on the Motorola MC68376 or of 50 ns on the Motorola PowerPC MPC555. The measurements are initiated by distinct events, such as the overflow of the TTP/C global time of a time master node (which occurs about every $\frac{1}{16}$ s in a cluster with a macro-tick length of about 1 μs) or the overflow of a full second. A time master node can configure the event that should initiate the offset measurements. Each of the time master nodes is configured to use the same initiating event. Figure 6.1 depicts a time master node with the time flow. A detailed description of the time master node implementation and the offset calculation can be found in [PB00].

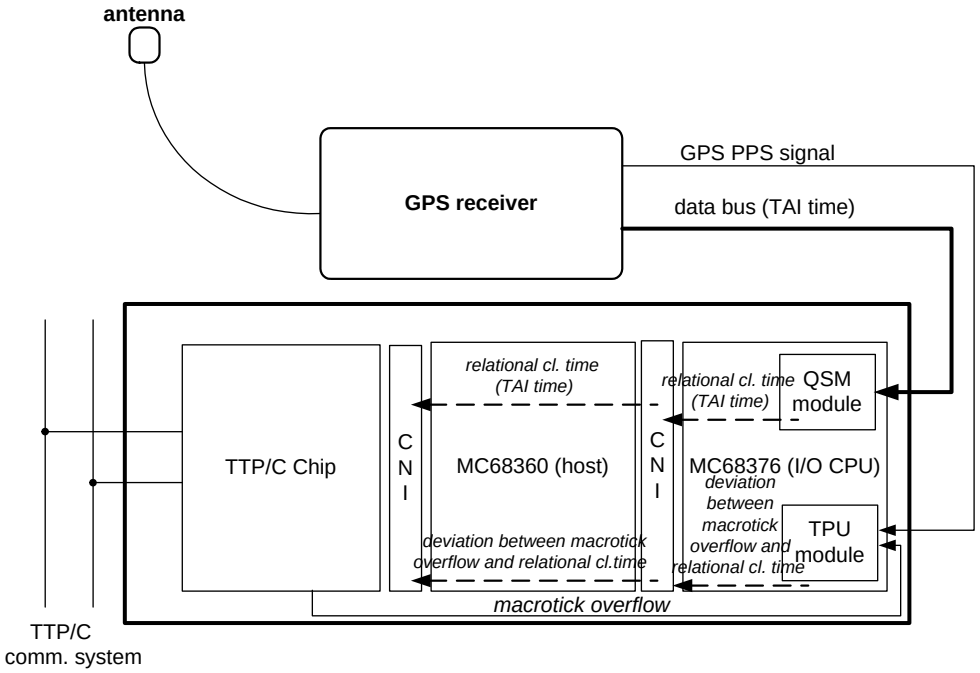


Figure 6.1: Implementation of a time master node with the Time Capturing Unit (TPU) and a GPS Receiver as external time source using a TTPNode

Sending of Offset Values, Agreement, and Calculation of Offset Value

Time master nodes send their offset values and relational clock values in their sending slots to all other nodes[1]. The TDMA round equals 1 ms in all imple-

[1]In the implementation using the TTA, time master nodes sends the relational clock time values to other nodes in each measurement interval, because the time representation used in the internal clock synchronization of *TTP/C* overflows at least once a second and the computational power of the TPU unit is low. As a consequence, it is more efficient to implement that nodes send the offset value of the TPU unit and the relational clock time

mentations. All nodes collect the values and perform the tasks described in Section 5.1.

Correction

Each node uses the common mode correction capability of *TTP/C* (see Section 2.3). Upon calculation of a new correction value (see Figure 5.1 and Section 5.1), each node divides the correction value of one measurement interval by the number of clock synchronization flags per measurement interval, denoted N_{syf}. The number of clock synchronization flags per measurement interval equals the number of TDMA rounds per measurement interval as one synchronization is performed per TDMA round. In order to avoid time consuming floating-point arithmetic, the division is performed using integer arithmetic resulting in an *integer part* and a *fractional part*. The fractional part is the remainder of an integer division. At the beginning of a TDMA round, each node adds the fractional part to a help variable, which is set to zero upon reception of a new correction value. When this adding produces an overflow, the node writes the integer part plus one into the common-mode-drift correction field otherwise it writes only the integer part in the common-mode-drift correction field. *TTP/C* uses the value of the common-mode-drift correction field for rate correction in the next TDMA round. The described mechanism is similar to the macrotick generation logic of *TTP/C*, which is described in [KHK+96, KKMS95], and allows the approximation of the correction of a fractional number each TDMA round by actually correcting a dynamically changing integer number.

6.3 Analysis of the Accuracy in the Time-Triggered Architecture

This section presents an analytical estimate of the accuracy of the external clock synchronization algorithm in the TTA in order to be able to interpret the measurements of the accuracy.

The accuracy of the global cluster time – using the described algorithm for external clock synchronization, the FTA algorithm [Kop97] for the internal clock synchronization, and external time sources as relational clock times – can be seen as a function of several parameters. The following list describes these parameters and presents values of implementations of the external clock synchronization using the TTA:

compared to an implementation where the TPU calculates and provides the offset value in a format where no overflows occur during the system's lifetime.

- **The stochastic part of the drift rate of the global cluster time** $\rho^{VT}_{\mathbf{stoch}}$. This drift rate originates in the environmental conditions and stochastic processes in the quartz crystals and digitalization errors and is typically approximately $5 \cdot 10^{-6}\ \frac{s}{s}$ or better.

- **The systematic part of the drift rate of the global cluster time** $\rho^{VT}_{\mathbf{sys}}$. This systematic error from the nominal frequency stems from the manufacturing process and aging of quartz crystals. ρ^{VT}_{sys} can be in the range of 10^{-7} to $1.86 \cdot 10^{-4} \frac{s}{s}$ [Kop01a], but is typically better than $10^{-4}\ \frac{s}{s}$.

- **The maximum drift rate of the relational clock times** $\bar{\rho}^{\mathcal{RCT}}$. If GPS is used as relational clock time the drift rate is less than $10^{-11}\ \frac{s}{s}$ [LAK99]. If another global cluster time is used as a relational clock time, the maximum drift rate is typically less than $10^{-4}\ \frac{s}{s}$.

- **The accuracy of the relational clock times** $\alpha^{\mathcal{RCT},\mathcal{T}}$. The accuracy of the GPS time is better than 5 ns with respect to the physical time standards TAI or UTC [LAK99]. The accuracy of other relational clocks depends on the synchronization quality of this relational clock to the reference clock time.

- **The reading error while obtaining the time of the relational clock times** $\mathbf{e}^{\mathcal{RCT}}$. The reading error for good commercially available GPS receivers is in the range of 50 ns to 100 ns [HS97]. The reading error when global times of other clusters work as relational clock times is considered to be 0.

- **The length of the offset measurement interval** $\mathbf{R_{measure}}$. The maximum possible rate of new clock values provided by relational clock times determines the length. It is in the range of 2^{-4} s to 1 s. Ideally, this should be a power of 2 of a full second, because then the external time can be measured at the same instant at different time master nodes as the least common multiple is always the rate with the largest value. Another reason for using a power of 2 is that the length of the interval can be represented in a computer system without introducing additional conversion errors.

- **The length of the external clock synchronization interval** $\mathbf{R_{ext.CS}}$. This can range from 2^{-1} s to 2^{6} s and depends on $R_{measure}$. A typical value is 1 s if $R_{measure}$ equals 2^{-4} s. The *history length* H is a multiple of the length of the time difference measurement interval that equals the length of the external clock synchronization interval, i.e., $R_{ext.CS} = H \cdot R_{measure}$; $H \geq 1$. Due to memory requirements, the history interval can be at most 256 for the presented implementation.

- **The granularity of the offset measuring unit $\mathbf{g_{measure}}$.** The granularity depends on the used hardware and introduces a digitalization error. The smallest and best granularity of the used offset measuring unit is 50 ns.

- **The granularity of the microtick $\mathbf{g_{\mu T}}$.** In *TTP/C*, a microtick is an integer multiply of a hardware tick. Microticks are used to build the virtual clocks. The granularity of the microtick is 50 ns in the current prototype version of the *TTP/C* controller [Kop98].

- **The time between the measurement of offset values and use of these values $\mathbf{d_{delay}}$.** It takes at most two TDMA rounds for sending the offset values to all other nodes. One TDMA round accounts for sending of all time master nodes of offset values and one for the time a node has to wait for its sending slot. A typical TDMA round length is 1 ms, consequently, $\mathbf{d_{delay}} = 2\ ms$.

- **The precision of the internal clock synchronization algorithm $\mathbf{\Pi}$.** The precision depends on many parameters; this is described in detail in [Kop99]. It is typically better than 0.5 μs.

Assuming that measurement and digitalization errors accumulate, Requirements 8 to 10 are satisfied, and the whole correction value can be corrected[2] in one $R_{measure}$, the analytical estimate of the accuracy of the global cluster time to reference time ($\alpha^{VT,\mathcal{T}}$) is as follows:

$$\begin{aligned}\alpha^{VT,\mathcal{T}} &= \alpha^{\mathcal{RCT},\mathcal{T}} + \Pi + g_{\mu T} + \\ &\quad \left(\rho^{VT}_{stoch} + 2 \cdot \bar{\rho}^{\mathcal{RCT}} + 2 \cdot \frac{g_{measure} + g_{\mu T} + e^{\mathcal{RCT}}}{R_{measure}}\right) \cdot \\ &\quad (R_{measure} + d_{delay})\end{aligned} \tag{6.1}$$

The first row of the right side of Equation 6.1 accounts for the inaccuracy of the external time source, the difference of the local node view of VT (precision Π) and a digitalization error of this view. The second and third row comprehend the drift rate of the relational clock times, the stochastic part of the drift rate and an additional drift rate due to errors in measurement and digitalization of the correction of ρ^{VT}_{stoch} and ρ^{VT}_{sys}. In detail: $\rho^{VT}_{stoch} \cdot (R_{measure} + d_{delay})$ accounts for the drift offset due to the stochastic part of the drift rate of the global time in the period of measuring and sending the values to other nodes; $2 \cdot \bar{\rho}^{\mathcal{RCT}} \cdot$

[2]There is just one case where a node cannot correct its whole correction value. This is when a node is re-synchronizing after a failure or after startup. In this case the accuracy value is not meaningful, as the system is not in synchronous operation.

$(R_{measure} + d_{delay})$ considers maximum drift rate changes and wrong estimates of the relational clock times; and $\left(2 \cdot \frac{g_{measure}+g_{\mu T}+e^{\mathcal{RCT}}}{R_{measure}}\right) \cdot (R_{measure} + d_{delay})$ accounts for the digitalization errors of measurements, the global cluster time, and the relational clock times that are wrongly corrected and/or estimated in the period of measuring and sending the values to other nodes.

Equation 6.1 only contains the interval $R_{measure}$ and not $R_{ext.CS}$, because a significant deviation of $\tilde{\rho}_{sys}^{VT,m}$ (which is the estimate of ρ_{sys}^{VT}) from ρ_{sys}^{VT} in the measurement interval of execution round n is considered as ρ_{stoch}^{VT} in the measurement interval of execution round n. ρ_{stoch}^{VT} is corrected in the measurement interval of execution round $n+1$. ρ_{sys}^{VT} does not influence the accuracy, because it is assumed that the average of the offset measurements of the external clock synchronization algorithm reflects only ρ_{sys}^{VT} and not ρ_{stoch}^{VT}.

ρ_{sys}^{VT} can change, however, if the set of nodes used in the internal clock synchronization changes. The maximum amount of this change $\rho^{\Delta_{nodes}}$ can be calculated using Equation 6.2.

$$\rho^{\Delta_{nodes}} = abs\left(max_{1\leq i\leq N}\left(\rho_{sys}^{\mathcal{VT}^i}\right) - min_{1\leq j\leq N}\left(\rho_{sys}^{\mathcal{VT}^j}\right)\right), \tag{6.2}$$

where $\mathcal{VT}^i$ and $\mathcal{VT}^j$, $1 \leq i, j \leq N$ and $i \neq j$, are used for building the global time of a cluster, $abs()$ calculates the absolute value of a number, $min()$ and $max()$ returns the minimum and maximum value of a set of values, respectively, N is the number of nodes. In systems with the same quartzes, $\rho^{\Delta_{nodes}}$ equals at most $2 \cdot \rho_{sys}^{VT}$.

If the drift rate can change by at most $\rho^{\Delta_{nodes}}$, the resulting analytical estimate of the accuracy of the global cluster time $\alpha^{VT,\mathcal{T}}_{\rho_{sys}^{VT}\ changes}$ is as follows:

$$\alpha^{VT,\mathcal{T}}_{\rho_{sys}^{VT}\ changes} = \alpha^{VT,\mathcal{T}} + \rho^{\Delta_{nodes}} \cdot (R_{measure} + d_{delay}) \tag{6.3}$$

Example: Using typical parameters – as listed above – and a $\rho^{\Delta_{nodes}}$ of $2 \cdot 10^{-4}\frac{s}{s}$ results in an the following estimates for external synchronization: $\alpha^{VT,\mathcal{T}} = 1.35\ \mu s$ and $\alpha^{VT,\mathcal{T}}_{\rho_{sys}^{VT}\ changes} = 14.3\ \mu s$.

6.4 Analysis of the Influence of the Algorithm on the Precision in the Time-Triggered Architecture

The following equations show the calculation of the influence of the external clock synchronization on the precision for the implementation. It is assumed

that the TDMA round duration ($\tau^{TDMA\ round}$), which determines R_{corr}, equals approximately 1 ms and the hardware clock time drift rate ($\bar{\rho}^{\mathcal{HT}}$) is at most $10^{-4}\frac{s}{s}$:

$$\bar{C} = \bar{\rho}^{corr} \cdot R_{corr} \tag{6.4}$$

Since $\bar{\rho}^{corr} > \bar{\rho}^{\mathcal{HT}}$ (Section 5.1):

$$\begin{aligned} \bar{C} &> \bar{\rho}^{\mathcal{HT}} \cdot \tau^{TDMA\ round} \\ &\approx 10^{-4} \cdot 10^{-3} = 10^{-7} = 100\ ns \end{aligned}$$

The correction term C must be a multiple of 50 ns, since the (nominal) hardware clock time frequency equals 20 MHz. The maximum correction term ($\bar{C}$) is at least 150 ns, because this is next multiple of 50 ns that is greater than 100 ns.

Using Equation 5.5, leads to the following influence of the external clock synchronization on the precision; $\bar{\epsilon}$ denotes this influence:

$$\begin{aligned} \bar{\epsilon} &= \bar{C} \cdot 2 \cdot \bar{\rho}^{\mathcal{HT}} \\ \bar{\epsilon} &= 150 \cdot 10^{-9} \cdot 2 \cdot 10^{-4} = 3 \cdot 10^{-11} s = 0.03\ ns \end{aligned}$$

Since the precision is in the order of 250 ns to 1 μs for TTP/C clusters, we conclude that the influence of the external clock synchronization on the precision of the global cluster time is negligible.

6.5 Analysis of Self-Stabilization in the Time-Triggered Architecture

In Section 5.4, we have argued that the multi-cluster clock synchronization algorithm is self-stabilizing. If the algorithm is based on the Time-Triggered Architecture (TTA), the TTA must be self-stabilizing in order that Requirements 2 to 10 can hold again after a transient fault effected the operation of the TTA.

In the TTA, two mechanisms exist upon which the multi-cluster clock synchronization base. These are the *startup algorithm* and the *group membership algorithm*. The startup algorithm brings a cluster from asynchronous to synchronous operation [SP02] and ensures that Requirements 2, 3, and 5 to 10 are guaranteed after a transient failure. Requirement 4 is outside the control of the TTA if relational clock times are external time sources. If the relational clock times are virtual clock times of nodes of other clusters, Requirement 4 is ensured by the startup and group membership algorithm.

The startup algorithm is self-stabilizing, because it follows the never-give-up (NGU) strategy and states of nodes do not have to be initialized [Ste02b]. In [Rus02], Rushby formally verifies that the TTA group membership algorithm ensures the consistency of participating nodes and that the group membership algorithm is proven to be self-stabilizing. As the startup and the group membership algorithm are self-stabilizing, Requirements 2 to 10 hold after a transient fault and the multi-cluster clock synchronization is also self-stabilizing as argued in Section 5.4.

6.6 Summary

This chapter described an implementation of the algorithm using the Time-Triggered Architecture. We first presented the goals of the implementation. Then, we analyzed the algorithm with respect to the achievable performance, in order to be able to draw conclusions of the influence of the parameters on the measured accuracy. We also analyzed the influence of the algorithm on the precision. The chapter concluded with an analysis of the fault-tolerant behavior of the algorithm.

Chapter 7

Experimental Evaluation

Experiments don't spring like Athena fully formed on the brow of Zeus.
They are painstaking[,] constructed by mortals
who usually get it wrong first time,
who require lengthy periods of exploration
before formulating a precise experimental question.

Paul Cohen

This chapter describes the different experiments that have been performed using the implementation of the algorithm on top of the Time-Triggered Architecture (TTA). This implementation has been described and analyzed in Chapter 5. Validation of the multi-cluster clock synchronization is performed by measurements of different cluster configurations. All experiments base on Requirements 8 and 9, that is the systematic part of the drift rate of the global time of a cluster is constant and the stochastic part is symmetrically distributed. This chapter first validates that Requirements 8 and 9 hold for the global time built by the internal clock synchronization algorithm used in the Time-Triggered Architecture, which is the Fault-Tolerant Average algorithm (see Section 2.3). As the global time is an abstract notion used for all synchronized virtual clock times of a cluster, the drift rate of the global time cannot be directly measured. Instead, the drift rate of one virtual clock time $\mathcal{VT}^i$, denoted $\rho^{\mathcal{VT}^i}$, that is synchronized with all other virtual clock times of the cluster is measured. Similarly, when evaluating the performance of the multi-cluster algorithm (in terms of accuracy), one virtual clock time of the global time of a cluster is taken as a representative for the global time of a

cluster. We use this approach and take the worst-case precision into account when analyzing the results. This worst-case precision has been measured and is also shown in one of the experiments.

7.1 Experimental Setup

All experiments use a cluster with 5 nodes and a TDMA round length of 1 ms. The cluster is configured to perform clock synchronization once in each TDMA round (in the 5^{th} slot). As described in Section 4.1, we distinguish between external and inter-cluster synchronization. In case of external synchronization, relational clock times are directly synchronized to a time standard. In case of an inter-cluster synchronization, relational clock times are not directly synchronized to a time standard. For the experiments, GPS is used as time standard, because it is commonly available, precise (the accuracy of receivers with respect to real time is approximately 50 ns), and GPS receivers are commercially available and, thus, cheap. For the evaluation of external synchronization, Requirement 7 is waived, because only one GPS receiver has been available for the experiments. Waiving Requirement 7 effects non-interference. This cannot be shown with only one time master node. In order to evaluate non-interference, the validation of non-interference has been subject of a different experiment and experimental setup, which is described in Section 7.8.

7.2 Prerequisites

The validity of Requirements 8 and 9 is a prerequisite that the multi-cluster clock synchronization algorithm can take advantage of the regularities of the drift rate of the global time of a cluster. These regularities can lead to the improvement of the accuracy of the clock synchronization compared to known clock synchronization techniques. This section evaluates the validity of Requirements 8 and 9 for *TTP/C* by measuring the drift rate of a global time of a cluster, denoted ρ^{VT}.

Virtual clocks base on hardware clocks, whose drift rates comprise a systematic and a stochastic part. The systematic part of hardware clocks is approximately constant and approximately 100 times larger than the stochastic part [Sch88]. The systematic part of the drift rate of hardware and virtual clocks can be eliminated, we say these clocks are calibrated. This section evaluates the drift rates of the virtual clock times before and after calibrating the virtual clocks using a reference clock.

Calibration of the virtual clocks has two advantages: First, the precision of a cluster is improved, as the drift rate of virtual clocks gets smaller and, subsequently, the drift offset gets smaller[1]. Secondly, as *TTP/C* uses only a subset of the virtual clocks of a cluster to build the global time of a cluster and the drift rates of different virtual clocks are different, the drift rate of the global time of a cluster changes when the set of nodes used for internal clock synchronization changes. Calibration decreases the change of the drift rate of the global time, when the set of nodes building the global time changes, as it decreases the amount of the differences between the drift rates of different virtual clocks.

Drift Rate of Global Time

Figure 7.1 shows a measured density distribution of the drift rate of virtual clock time i, denoted $\rho^{\mathcal{VT}^i}$, (over a period of approximately 8 hours) while virtual clock i is synchronized to other virtual clocks of a cluster; node i is one of the cluster nodes. This is a good indication for drift rate of the global time of the cluster, denoted ρ^{VT}. From these measurements, which are depicted in detail in Table B.1 in Appendix B, it can be concluded that $\rho^{VT}_{sys} \approx 4.2 \cdot 10^{-5}\ \frac{s}{s}$, $\rho^{VT}_{stoch} \lessapprox 6 \cdot 10^{-6}\ \frac{s}{s}$. The distribution is approximately symmetrical and, thus, provides evidence that Requirements 8 and 9 hold for the TTA.

Calibrated Clocks

Identifying situations for *TTP/C* where the systematic part of the drift rate of the global time of a cluster, denoted ρ^{VT}_{sys}, changes and where these changes can be controlled enables the measurements of the consequences of a change of the systematic part of the drift rate of the global time. This section describes how the the systematic part of the drift rate of the global time can be changed by calibrating the hardware clocks of nodes.

In *TTP/C*, the set of nodes whose virtual clock times are used for internal clock synchronization can change. This can happen due to several reasons, such as a node encounters a failure or a node decides to stop computational activities. In *TTP/C*, the drift rate of virtual clock times can be adjusted. This enables the minimization of the differences between the drift rates of different virtual clocks of a clusters. Calibration of virtual clock times leads to a minimization of the changes of the systematic part of the drift rate of the global time, if the set of nodes used for internal clock synchronization changes.

[1]Please note that the precision is only improved, if virtual clocks are not nearly perfect before calibration and, thus, can be improved by calibration, as is the case in a realistic setup.

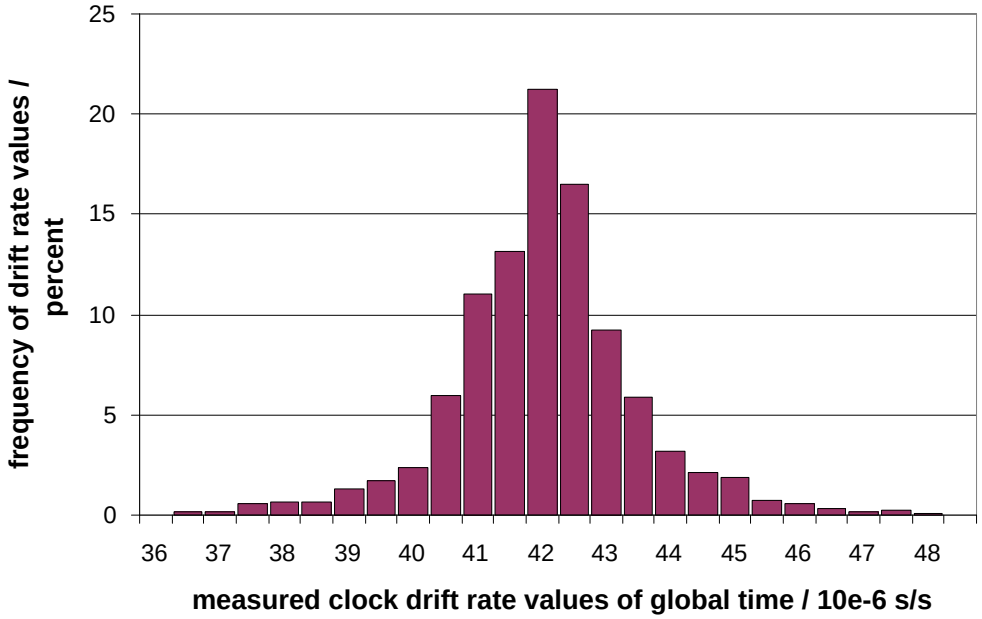


Figure 7.1: Density distribution of the drift rate of a virtual clock synchronized with all virtual clocks of a cluster, which is a good approximation of the density distribution of the drift rate of the global time of the cluster

In order to adjust the systematic part of the drift rate of virtual cluster times, the actual systematic part of the drift rate of each node has been measured with respect to a relational clock time ($\mathcal{RCT}$), which is closely synchronized to real time (GPS Time). Then, the systematic part of the drift rate of each node is set to the minimum drift rate that *TTP/C* allows. This minimum systematic part of the drift rate after calibration depends on several parameters, is due to finite representations of scaling factors, and is explained in more detail in [KKMS95]. Minimizing the systematic part of the drift rate of the virtual times of a cluster with respect to $\mathcal{RCT}$ minimizes the systematic part of the drift rate of the global time by the same amount. For the hardware under test (which are *TTP/C* clusters), the maximum systematic part of the drift rate of the global time after the calibration is $800 \cdot 10^{-9} \frac{s}{s}$. We call a cluster where the systematic part of the drift rate of its nodes has been calibrated w.r.t. an accurate external time source a *calibrated cluster* otherwise *not calibrated cluster*. Figure 7.2 and 7.3 depict the deviation of the virtual cluster times of nodes w.r.t. the relational clock time ($\mathcal{RCT}$) without and with calibrating, respectively (please, note the different scales). As the systematic part of the drift rate of the global time of a cluster is smaller than the largest value of the drift rate of any of the virtual clocks of a cluster, the systematic part of the drift rate of the global time of the calibrated cluster, denoted $\rho_{sys}^{VT,\ calibrated}$, is at most $0.44 \cdot 10^{-6} \frac{s}{s}$, while the drift rate of the global time of the not cali-

brated cluster, denoted $\rho_{sys}^{VT,\ not\ \ calibrated}$, is at most $16 \cdot 10^{-6} \frac{s}{s}$ for the measured configurations.

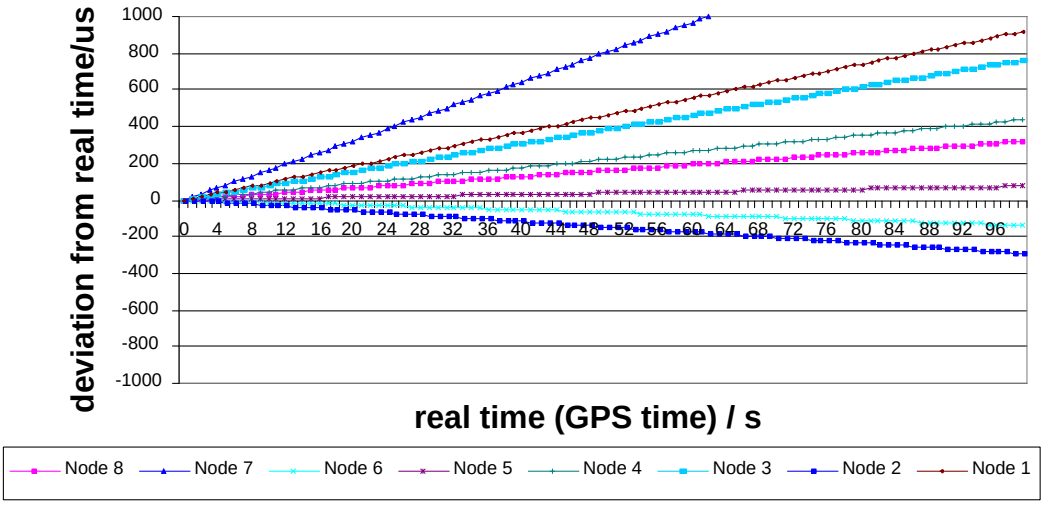


Figure 7.2: Deviation of virtual clock times from relational clock time (this is closely synchronized to real time, that is GPS time), without calibrating (i.e., with nominal) drift rate of the virtual clocks (of the nodes). The nodes do not perform any synchronization.

The next sections describe the different experiments. Each section is divided into four distinct parts. The first part describes the goal of the experiment and specialities of the setup. The second part discusses the expected outcome or describes the hypothesis of the experiment. The third part presents the measurements and results. The last part discusses the results and compares them with the expected outcome. Unless otherwise defined, each experiment uses the configurations described at the beginning of this chapter.

7.3 Long-Term Evaluation of Accuracy

This experiment evaluates the accuracy and long-term stability of the external clock synchronization algorithm for inter-cluster and external synchronization. For the whole measurement period, ρ_{sys}^{VT} is not changed.

The deviation values of the external clock synchronization algorithm are measured. The deviation values are the differences between different virtual clocks times and relational clock times before the correction and, thus, before convergence of the different virtual clock times starts. Consequently, these deviation values are the maximum difference between the virtual clock times

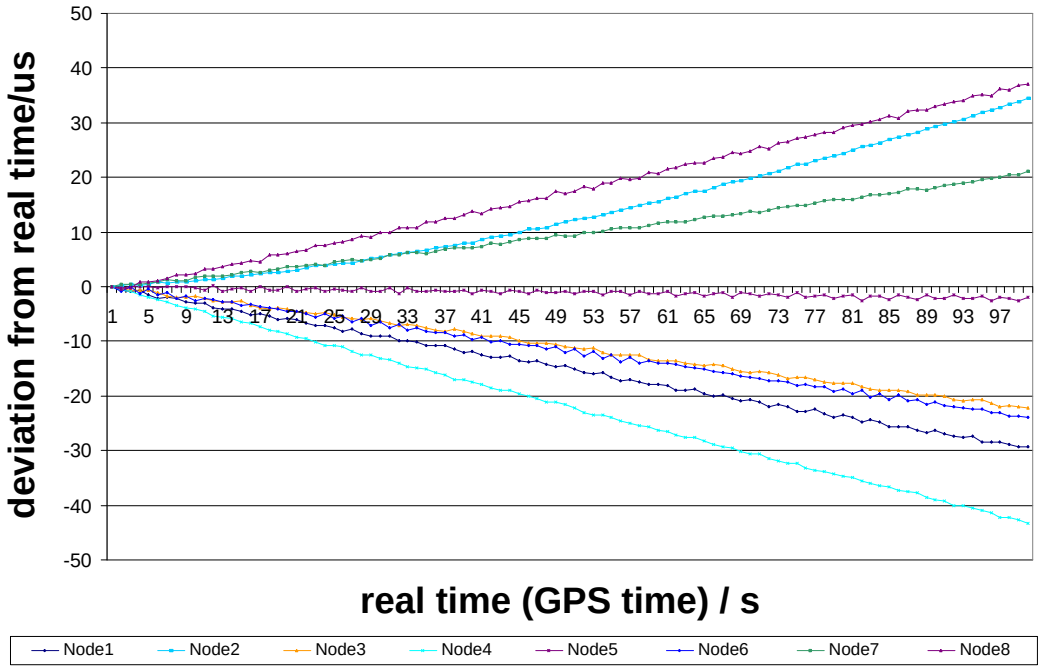


Figure 7.3: Deviation of virtual clock times from relational clock time (this is closely synchronized to real time (GPS time), after calibration of the drift rate. The nodes do not perform any synchronization.

and the relational clock times. As the period between the measurement of the deviation values and the start of the correction is much smaller[2] than the measurement interval, the drift offset in this period is negligible and not taken into account, although it could slightly increase (that is worsen) the accuracy. This experiment uses the TTPpowernode evaluation board for time master nodes. For the TTPpowernode evaluation board, the deviation values are multiples of 50 ns due to the hardware configuration and the frequency of the clock generator of the measurement unit.

Expected Outcome and Hypotheses

For inter-cluster and external synchronization, the frequency diagram of the measured deviation values should approximately form a symmetrical distribution around mean 0 s. For external synchronization, the maximum of the absolute amount of the measured deviation values should be less than 6.37 μs, because the worst-case estimate of the accuracy $\alpha^{VT,\mathcal{T}}$ is 6.97 μs according to

[2]The measurement interval is in the order of 1 second, while the period is in the order of 2 ms, which comprises the time for sending the values and calculation of the correction value.

Equation 6.1 and the following parameter values are assumed: $\Pi = 0.5\ \mu s$, $\alpha^{\mathcal{RCT},\mathcal{T}} = 5\ ns$, and $e^{\mathcal{RCT}} = 100\ ns$. The precision of the cluster is subtracted from the estimate, because the deviation values are only measured at one time master node and all nodes of a cluster are synchronized to each other within the precision. The substraction of the other two parameters accounts for the synchronization inaccuracy of the relational clock time to the reference clock time (and, consequently, real time) and digitalization errors introduced by the acquisition of representation of a time standard.

Similarly, the maximum deviation values of inter-cluster synchronization should not exceed 850 ns. The worst-case estimate of the accuracy is 1.35 μs according to Equation 6.1. The following parameters are assumed for inter-cluster synchronization: $\Pi = 0.5\ \mu s$, $\alpha^{\mathcal{RCT},\mathcal{T}} = 0\ ns$, and $e^{\mathcal{RCT}} = 0\ ns$. As the relational clock time is not synchronized to real time and inter-cluster synchronization is not concerned with direct synchronization to real time, the accuracy with respect to real time is of no concern and $\alpha^{\mathcal{RCT},\mathcal{T}}$ and $e^{\mathcal{RCT}}$ are set to 0. Again, the precision of the cluster is subtracted from the estimate, because the deviation values are only measured at one time master node and the virtual clock times of all nodes (and consequently the virtual clock times of time master nodes) of a cluster can be at most the value of the precision apart.

Experiment Results

Figure 7.4 depicts the measured deviation values of external synchronization to a relational clock time, denoted $\mathcal{RCT}$, which is synchronized to GPS time. The results represent the measurements over a period of 28 hours. Evaluation of the data, which is depicted in detail in Table B.2 in Appendix B, shows a maximum deviation[3] $of 850$ ns, leading to $\alpha^{VT,\mathcal{T}} = 1.45\ \mu s$ (assuming $\Pi = 0.5\ \mu s$, $\alpha^{\mathcal{RCT},\mathcal{T}} = 5\ ns$, and $e^{\mathcal{RCT}} = 100\ ns$).

Figure 7.5 depicts the measured deviation values of inter-cluster synchronization over a period of 2 hours, which equals an measurement period of approximately 10^5 measured values. As can be seen in Table B.3 in Appendix B, the maximum deviation[4] during this period is 450 ns, which equals an accuracy $\alpha^{VT,\mathcal{RCT}}$ of 950 ns (assuming $\Pi = 0.5\ \mu s$).

[3] 850 ns correspond to 17 TPU ticks, each of length 50 ns. 17 TPU ticks is the largest measured deviation value as can be seen in Table B.2 in Appendix B.

[4] 450 ns correspond to 9 TPU ticks, each of length 50 ns. 9 TPU ticks is the largest measured deviation value in Table B.3 in Appendix B.

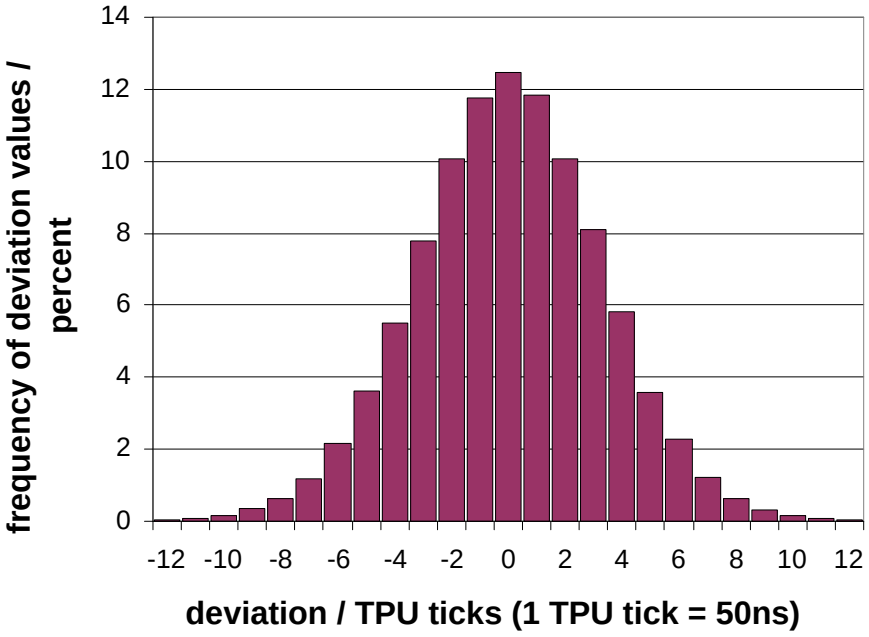


Figure 7.4: External synchronization: deviation of virtual clock time from relational clock time ($R_{ext.CS} = 16\ s$, $R_{measure} = 1\ s$, $g_{measure} = 1$ TPU tick length $= 50\ ns$)

Discussion of Results

For external synchronization, the worst-case estimate of the accuracy is 6.97 μs, while the measured accuracy equals 1.45 μs. Thus, the measured accuracy value is only 21 % of the worst-case estimate or 5.52 μs smaller than the estimated value. For inter-cluster synchronization, the worst-case estimate of the accuracy equals 1.35 μs and the measured accuracy is 950 ns. The measured accuracy value is 70 % of the worst-case estimate of the accuracy or 400 ns smaller than the estimated value.

The experiments of this section show that the algorithm performs as expected, because, first, the measured deviation values approximately form a symmetrical distribution around a mean of 0 s, and, secondly, the maximum amount of the measured deviation values are smaller than the estimated worst-case accuracy values. The cluster that is used for the experiments is not calibrated and the systematic part of the drift rate is approximately $16 \cdot 10^{-6}\ \frac{s}{s}$ (see Section 7.2). If the systematic part of the drift rate of the global time were not corrected, the mean of the distribution would be approximately 1.6 μs for the external synchronization and approximately 0.1 μs for inter-cluster synchronization, because the measurement interval is 1 s and $\frac{1}{16}$ s, respectively.

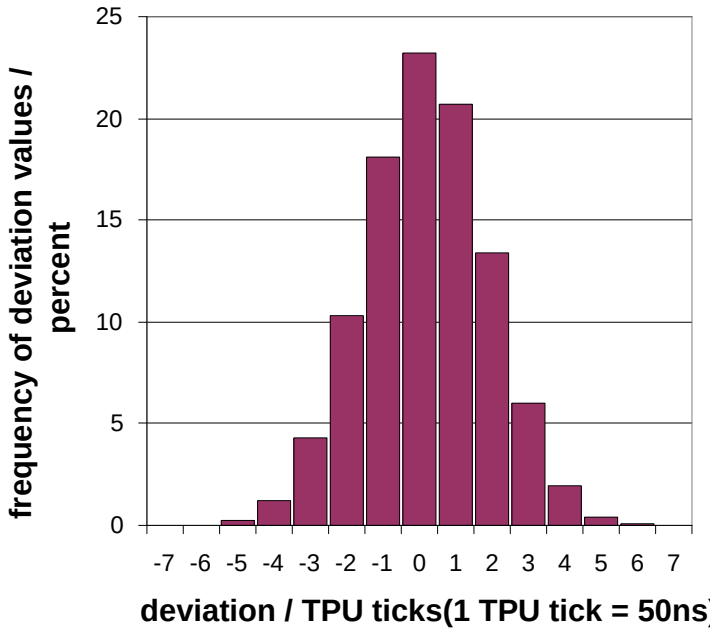


Figure 7.5: Inter-cluster synchronization: deviation of virtual clock time from relational clock time ($R_{ext.CS} = 1\ s$, $R_{measure} = \frac{1}{16}\ s$, $g_{measure} = 50\ ns$)

7.4 Independence of the Accuracy from the Value of the Systematic Part of the Drift Rate

This experiment yields at showing the independence of the achieved accuracy from the value of the systematic part of the drift rate of the global time, denoted ρ_{sys}^{VT}. For this, the deviation values of a calibrated and a not calibrated cluster are measured.

The values of the systematic part of the drift rate of the global time are significantly different for a calibrated and a not calibrated cluster. The systematic part of the drift rate is approximately $0.44 \cdot 10^{-6} \frac{s}{s}$ for the calibrated cluster and approximately $16 \cdot 10^{-6} \frac{s}{s}$ for the not calibrated cluster, as shown in Section 7.2. This experiment uses the property of different systematic parts of the drift rates for the calibrated and not calibrated clusters to evaluate whether the value of the systematic part of the drift rate has an effect on the performance of the multi-cluster clock synchronization algorithm or not. Time master nodes are implemented using the TTPnode evaluation boards. Consequently, the deviation values are integer multiples of 190.7 ns.

Expected Outcome and Hypotheses

As discussed in Section 6.3 and manifests in Equation 6.1, the systematic part of the drift rate does not have an effect on the achieved accuracy. Consequently, we expect the deviation values of the calibrated and not calibrated cluster to be in the same range. The frequency diagrams of both configurations should follow the same symmetrical distribution function around a mean of 0 s.

Experiment Results

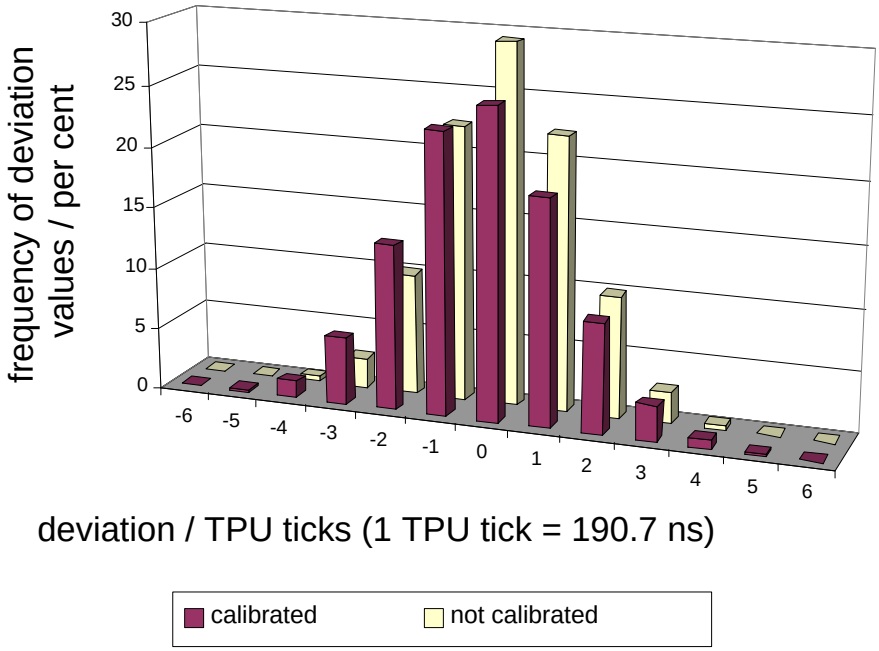


Figure 7.6: External Synchronization: deviation of virtual clock time from relational clock time ($R_{ext.CS} = 16\ s$, $R_{measure} = 1\ s$, $g_{measure} = 190.7\ ns$)

Figure 7.6 shows the frequency diagrams of the deviation values of the calibrated and not calibrated cluster.

Discussion of Results

Figure 7.6 and Table B.4 in Appendix B show that the distribution of the deviation values of the not calibrated cluster does not deviate significantly from the distribution of the deviation values of the calibrated cluster. As described above, the systematic part of the drift rate of the global time is approximately $0.44 \cdot 10^{-6} \frac{s}{s}$ for the calibrated cluster and approximately $16 \cdot 10^{-6} \frac{s}{s}$ for the not

calibrated cluster. Since the values of the systematic parts of the drift rates of both cluster configurations are different, but the achieved accuracy is the same and follows approximately the same distribution with a mean of approximately 0 s, we conclude that the used algorithm can correct a systematic part of the drift rate well. As the value of the systematic part of the drift rate of the not calibrated cluster is approximately 36 times larger than the systematic part of the drift rate of the calibrated cluster, we conclude that the actual value of the systematic part of the drift rate does not effect the quality of synchronization.

7.5 Change of the Systematic Part of the Drift Rate of the Global Time

This experiment aims at observing the effects of a change of the drift rate of the global time, denoted ρ_{sys}^{VT}. For this experiment, ρ_{sys}^{VT} is changed at a certain point in time. The deviation values are measured before and after the change.

As described in Section 7.2, the values of the systematic part of the drift rate of the global time are significantly different for a calibrated and a not calibrated cluster. Moreover, the drift rate of the global time stays nearly the same when the set of nodes used for internal clock synchronization changes, because the differences between the virtual clock times of a calibrated cluster are nearly the same. For a not calibrated cluster, the drift rate can change by an amount of up to $\rho^{\Delta_{nodes}}$ (see Section 6.3). These properties of the calibrated and not calibrated clusters are used to evaluate whether a change of the systematic part of the drift rate of the global time can have an effect. Please note, that the constant systematic part of the drift rate in a specified period of time (Requirement 8) is valid before and after the change. This experiment is performed using the TTPnode evaluation boards for time master nodes. For TTPnode evaluation boards, the deviation values are integer multiples of 190.7 ns.

Expected Outcome and Hypotheses

Using Equation 6.2 and the measured values depicted in Figure 7.2 and 7.3, leads to the following values of the change $\rho_{not\ calibrated}^{\Delta_{nodes}} = 19 \cdot 10^{-6} \frac{s}{s}$ and $\rho_{calibrated}^{\Delta_{nodes}} = 0.72 \cdot 10^{-6} \frac{s}{s}$. Using Equation 6.3, the measured $\rho^{\Delta_{nodes}}$, and the parameters $R_{measure} = 1\ s$, $R_{ext.CS} = 32\ s$, $g_{measure} = 190.7\ ns$, $\Pi = 0.5\ \mu s$,[5] $\alpha^{\mathcal{RCT},\mathcal{T}} = 5\ ns$, and $e^{\mathcal{RCT}} = 100\ ns$ for this external synchronization to calculate the estimate, gives the following results: $\alpha_{\rho_{sys}^{VT}\ changes}^{VT,\mathcal{T}} = 7.97\ \mu s$ and a maximum

[5]Here, we disregard the effect that the calibration can improve the precision of a cluster.

deviation value of 7.37 μs for the calibrated cluster and $\alpha^{VT,\mathcal{T}}_{\rho^{VT}_{sys}\ changes} = 26.3\ \mu s$ and a maximum deviation value of 25.7 μs for the not calibrated cluster.

The diagram depicting the deviation values with a changing value of ρ^{VT}_{sys} should show that – if ρ^{VT}_{sys} changes at t – at latest at $t + 2 \cdot R_{ext.CS}$ the deviation values should be again in the value range in which they were before t.

Experiment Results

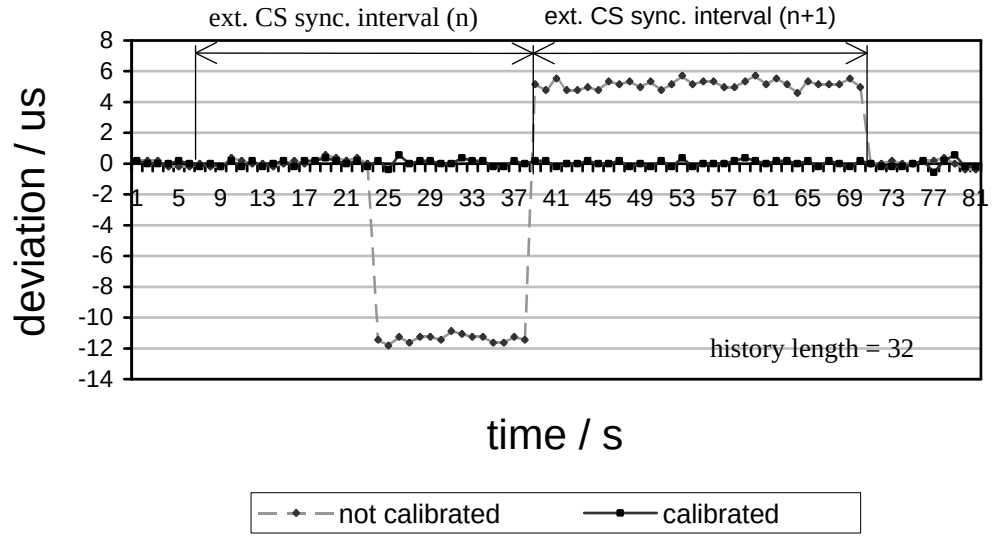


Figure 7.7: External Synchronization: deviation after a change in the set of nodes used for internal clock synchronization ($R_{ext.CS} = 32\ s$, $R_{measure} = 1\ s$, $g_{measure} = 190.7\ ns$)

Figure 7.7 depicts the effect of a change in the set of nodes used for internal clock synchronization that results in a change of ρ^{VT}. In this case at time 24, ρ^{VT}_{sys} changes. Analysis of the measured data, which is shown in Table B.5 in Appendix B, shows that the deviation is at most 800 ns ($\alpha^{VT,\mathcal{T}} = 1.3\ \mu s$) at the calibrated cluster. At the not calibrated cluster, the deviation is at most 12 μs ($\alpha^{VT,\mathcal{T}} = 12.6\ \mu s$).

Discussion of Results

When a cluster is calibrated the systematic part of the drift rate of the global time (ρ^{VT}_{sys}) does not significantly change (see Section 7.2), when the set of nodes used for internal clock synchronization changes. In contrasts, ρ^{VT}_{sys} changes significantly, when a cluster is not calibrated. This effect is used to evaluate

the effect of a change of ρ_{sys}^{VT}. Figure 7.7 and Table B.5 in Appendix B depict the deviation values of a calibrated and a not calibrated cluster where the set of nodes used for internal clock synchronization changes at time 24.

As one can see in the figures and tables, a change in the set of nodes used in the internal clock synchronization does not significantly affect ρ_{sys}^{VT} and, consequently, the measured deviation values and $\alpha^{VT,\mathcal{T}}$ do not change significantly. At the not calibrated cluster, ρ_{sys}^{VT} is significantly changed and, consequently, the deviation values significantly change for a period that is at most $2 \cdot R_{ext.CS}$ long.

Figure 7.7 shows that the external clock synchronization algorithm uses the calculated average of all medians of the set of deviation values of external clock synchronization interval n, denoted $R_{ext.CS}^{n}$, for the correction in $R_{ext.CS}^{n+1}$. It also shows that it takes at most $2 \cdot R_{ext.CS}$ until the global time of the cluster (VT) is synchronized again to a level where one cannot realize the change in ρ_{sys}^{VT} anymore. One duration $R_{ext.CS}$ stems from the abrupt change of ρ_{sys}^{VT}, the other one from the fact that the average of the measured deviations of one external clock synchronization interval is used for the next one.

We conclude from these observations and analysis of the algorithm that – if the set of nodes used for internal clock synchronization may change – calibrating the local drift rates of the nodes results in a significant improvement of the accuracy of the global time $\alpha^{VT,\mathcal{T}}_{\rho_{sys}^{VT}\ changes}$ compared to the accuracy of a cluster whose nodes have not been calibrated. This improvement is proportional to the maximum change of ρ_{sys}^{VT} if different sets of nodes are used in internal clock synchronization.

7.6 Changing History Length

This experiment aims at measuring the effects of different values for H or, in other words, the effects of a varying external clock synchronization interval ($R_{ext.CS}$) for a given measurement interval ($R_{measure}$). The deviation values for different values of H are measured.

This experiment is performed on a cluster that uses TTPnode evaluation boards as time master nodes. Thus, the deviation values are integer multiples of 50 ns.

Expected Outcome and Hypotheses

The maximum deviation values are expected to get smaller for larger values of H, because the approximation of ρ_{sys}^{VT} will be more precise for larger values

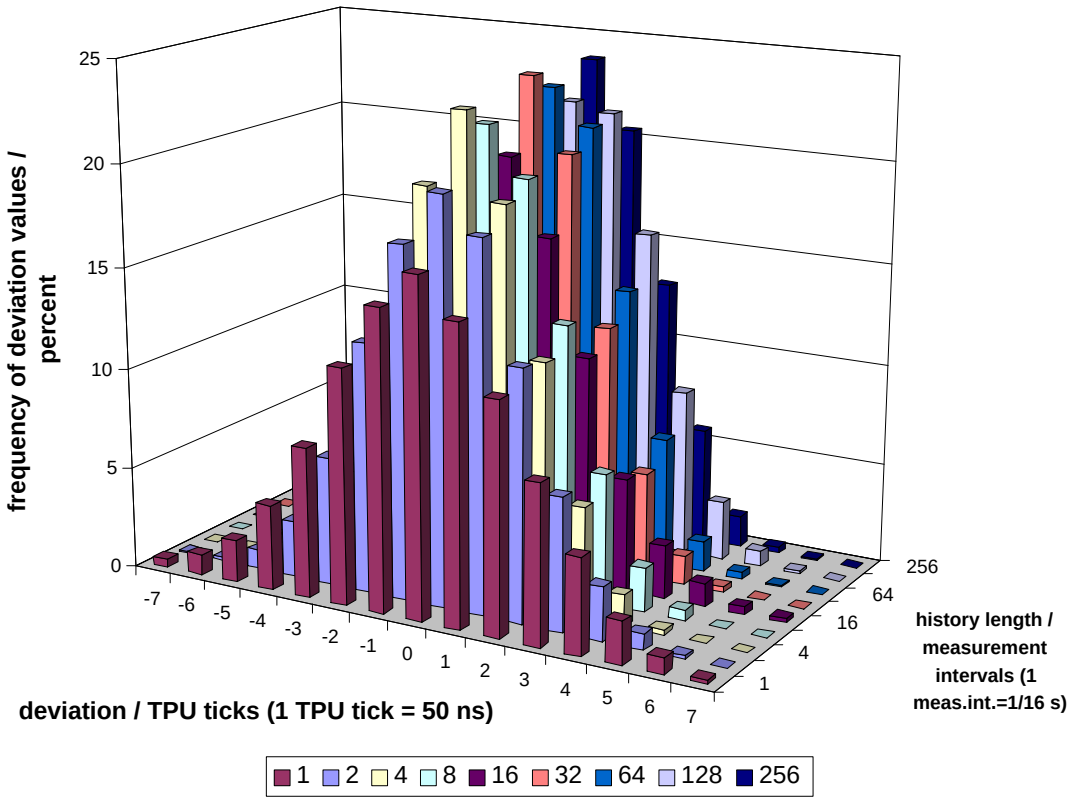


Figure 7.8: Inter-cluster synchronization: deviation of virtual cluster time from relational clock time with changing history length ($R_{measure} = 1/16\ s$, $g_{measure} = 50\ ns$)

of H and the external clock synchronization cannot "distinguish" between the effects of ρ_{sys}^{VT} and ρ_{stoch}^{VT} for small values of H. Once the value of H is sufficiently large for a precise approximation of the systematic part of the drift rate, the accuracy value will not get much better. The values of H that are "sufficiently large" depend on the density distribution of the stochastic part of the drift rate of the global time.

Experiment Results

Figure 7.8 and Table B.6 in Appendix B depict the measured deviations of inter-cluster synchronization. Figure 7.9 and Table B.7 in Appendix B show the measured deviations of external synchronization. In Figures 7.8 and 7.9, *one* color depicts the distribution of the frequency of the measured deviation values for a *constant* history length. The distribution with the shortest history length ($H = 1$) can be seen in the front, the distribution with longest history length ($H = 256$) is in back, other distributions are in between with distributions for

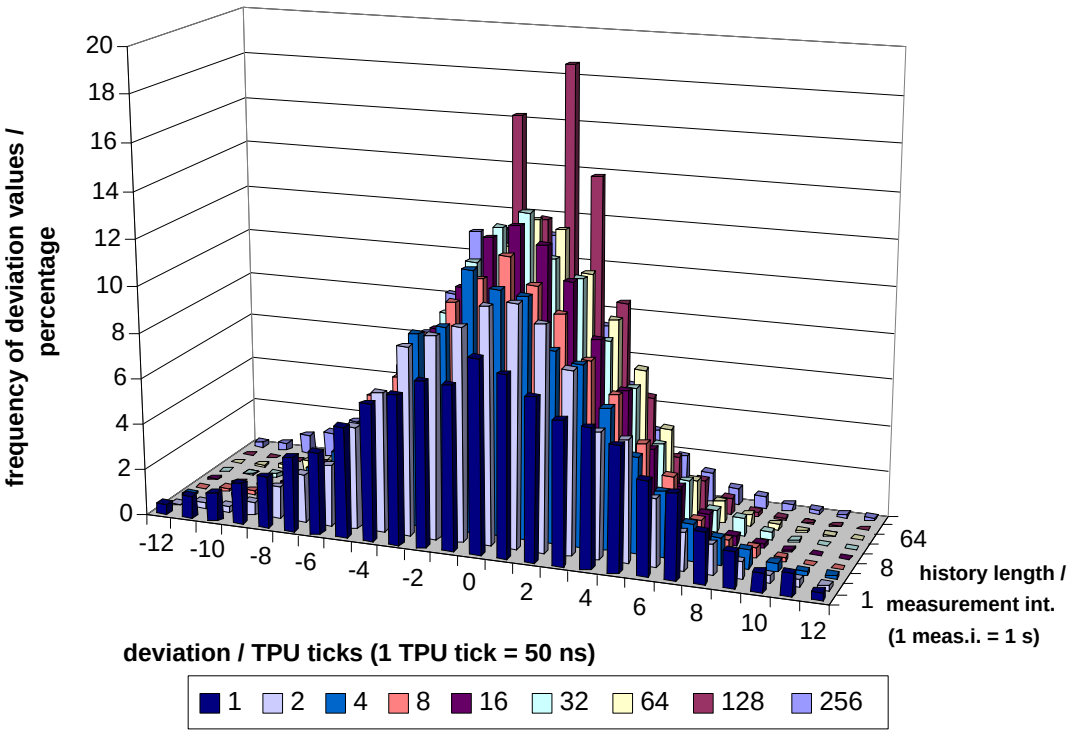


Figure 7.9: External synchronization: deviation of virtual clock time from relational clock time with changing history length ($R_{measure} = 1\ s$, $g_{measure} = 50\ ns$)

longer history lengths being "behind" shorter ones.

Analysis of the data shows that the best deviation values for the inter-cluster synchronization range from 300 to 500 ns depending on the history length. Assuming $\Pi = 0.5\ \mu s$, this corresponds to an accuracy of 800 ns to 1 μs. The best values for the external synchronization range from 500 ns to 1.2 μs, which equals an accuracy of 1.1 to 1.8 μs.

As can be seen in Tables B.6 and B.7 in Appendix B, the accuracy values get smaller (that is better) for increasing history lengths up to a certain bound. From this bound on, which equals 256 for the presented configurations, the accuracy values get significantly worse.

Discussion of Results

For external synchronization, the distribution of the deviation values where the history length equals 128 is not as regular as other distributions, as can be seen when comparing the second column from the right ($H = 128$) of Table B.7 in Appendix B with other columns of the same table. The spikes in the frequency

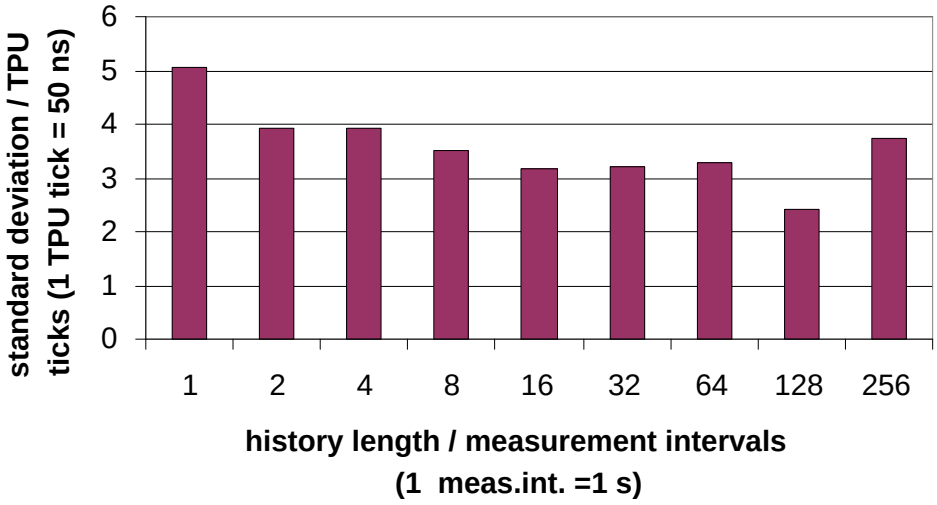


Figure 7.10: External synchronization: standard deviations of results presented in Figure 7.9.

diagram of the external synchronization (Figure 7.9) also depict this irregularity. These irregularities are due to the values $g_{measure}$ and $g_{\mu T}$ that represent the estimate of the systematic part of the drift rate of the global time. In this case, the estimate of the systematic part is "at the border" between two digital representations and, thus, the digitalization error has a major effect on the distribution. In order to provide evidence for this explanation, an analysis of the standard deviations of the data for the different history lengths is performed and presented in Figure 7.10 and Table B.8 in Appendix B. An analysis of the standard deviations shows that the standard deviation of the deviation values measured with external synchronization with history length 128 is even better than the ones measured with other history lengths. Consequently, the above explanation is a valid.

Moreover, Figure 7.10 shows that the standard deviation of synchronization with small history lengths (1 to 4) is significantly higher than the ones measured with history lengths ranging from 16 to 128. Interestingly, for inter-cluster and external synchronization the accuracy is best, when the history length is in the middle of the variations (4 to 32). We expected the deviation values to get smaller or at least remain equal for larger values of H. The accuracy gets worse for large values of H due to two reasons: First, digitization errors that approximate ρ_{sys}^{VT} have more influence if H is larger. Secondly, if the history length is long (64 to 256), the assumption of an approximately constant systematic part of the drift rate (Requirement 8) may not be an adequate assumption anymore and the linear approximation of the long-term drift rate of the algorithm is not optimal anymore.

While we can say that the optimal history lengths for these cluster configurations range from 16 to 128, generally, the optimal length depends on the properties of the nodes and the cluster configurations. The value of the history length should be small, because then the required space in memory necessary for storing the measured offset values is also small. Yet, the history length should be as large as necessary for being able to "well estimate" the systematic part of the drift rate. The algorithm "well estimates" the systematic part of the drift rate, if the error that can be made when calculating the average of the H offset values is smaller than the digitalization error.

7.7 Changing Measurement Parameters

This experiment examines the influence of the granularity of the offset measuring unit, denoted $g_{measure}$, and the length of the measurement interval, denoted $R_{measure}$, on the accuracy. The deviation values are measured for different measurement parameters.

This experiment is performed on a cluster that uses TTPpowernode evaluation boards as time master nodes. Thus, the deviation values are integer multiples of 50 ns.

Expected Outcome and Hypotheses

We expect the accuracy to be directly proportional to $R_{measure}$, because – given a constant ρ_{sys}^{VT} (Requirement 8) – the drift offset of the global time will be less in a shorter period. The smaller the measuring granularity, the better ρ_{sys}^{VT} can be measured and estimated, and the better will the accuracy be. At a certain point, decreasing the measurement granularity will not lead to an improvement of the achieved accuracy, because stochastic effects will dominate the accuracy.

Experiment Results

Figure 7.11 and Table B.9 in Appendix B show the standard deviations of the deviation values at different configurations. Figure 7.12 and Table B.10 in Appendix B depict the range of the different configurations. The *range* is the difference between the maximum and minimum deviation value and is an indicator of the accuracy ($\alpha^{VT,\mathcal{RCT}} \approx \frac{range}{2} + \Pi$). Different colors in Figures 7.11 and 7.12 represent different value intervals.

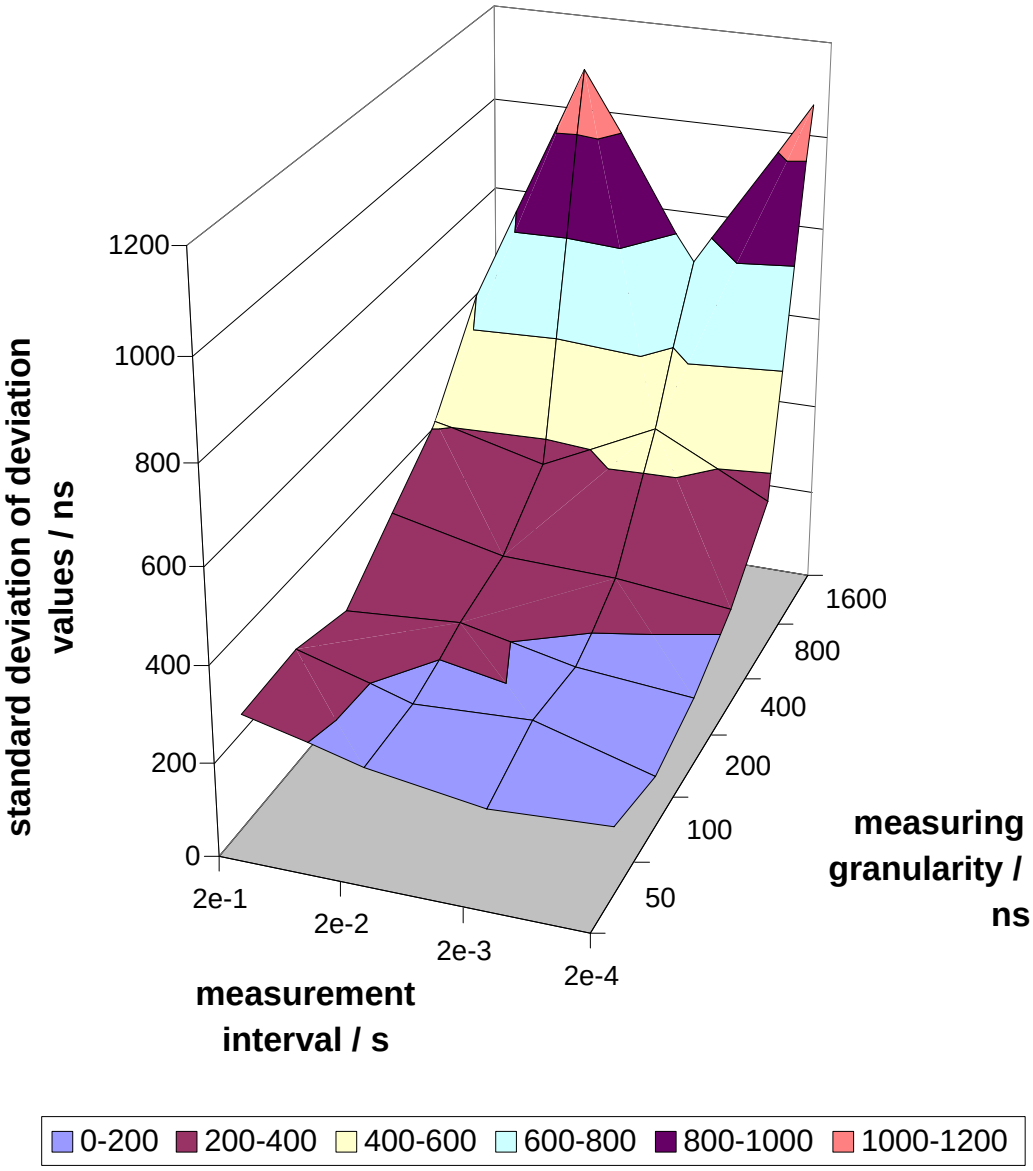


Figure 7.11: Inter-cluster synchronization: standard deviation of deviation values. $R_{measure}$ and $g_{measure}$ varying, $H = 16$.

Discussion of Results

As can be seen in Figures 7.11 and 7.12 and in Tables B.9 and B.10 in Appendix B and as expected, the standard deviations and ranges are increasing with increasing granularity of the offset measuring unit (digitalization error), denoted $g_{measure}$.

For measuring granularities ($g_{measure}$) smaller than or equal to 400 ns, the

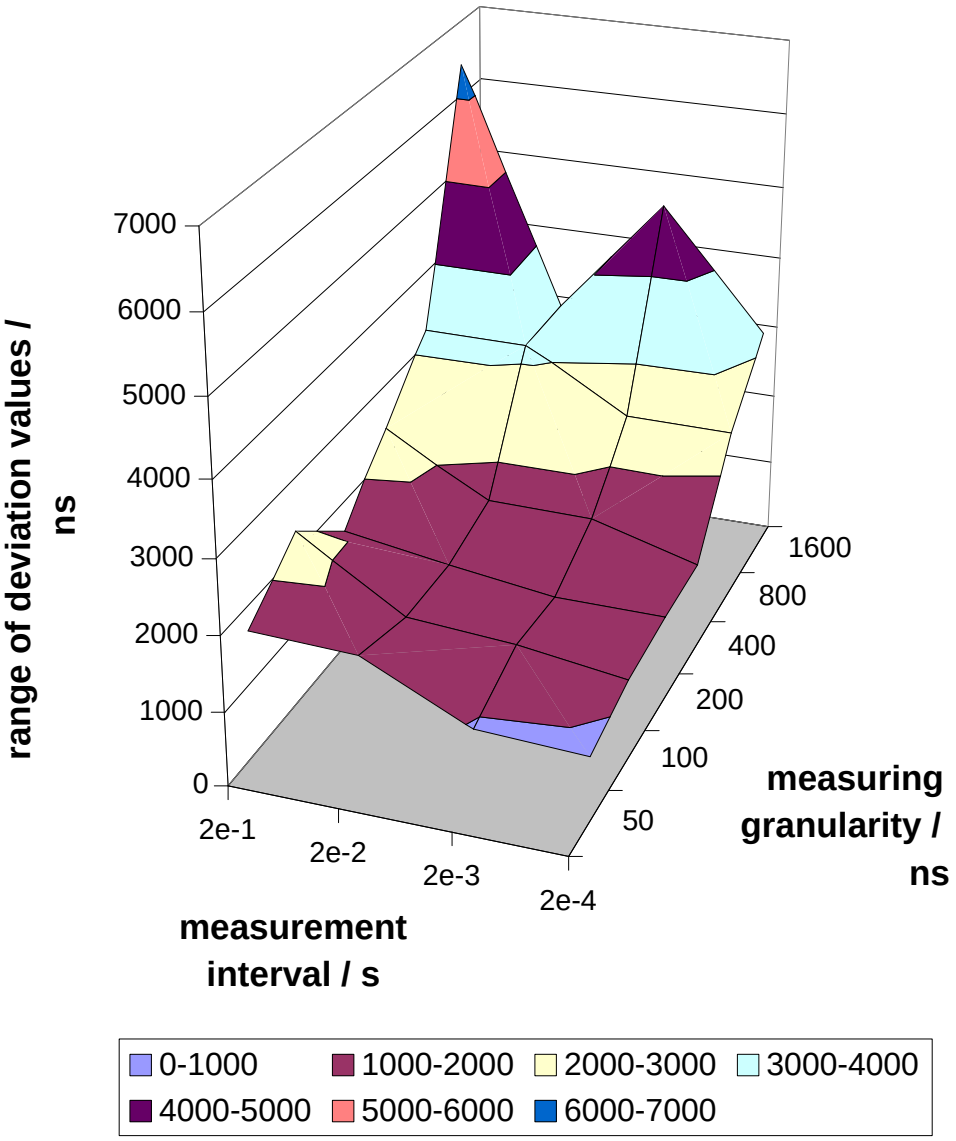


Figure 7.12: Inter-cluster synchronization: range of deviation values. $R_{measure}$ and $g_{measure}$ varying, $H = 16$.

standard deviation values and the accuracy values increase (that is worsen) for increasing measurement intervals. This increase of deviation and accuracy values is due to larger drift offset values. For measuring granularities greater than 400 ns, there is no monotonically increasing relation between the accuracy and the measurement interval, because digitalization errors dominate the outcome of the external clock synchronization algorithm and, thus, the achieved accuracy and stochastic effects play a major role in the achieved accuracy. This effect leads to the "cones" that can be seen in Figure 7.11 and 7.12

7.8 Non-Interference

This experiment aims at measuring the influence of the external clock synchronization algorithm on the precision of a cluster. For this, the precision of a cluster is measured once when the external clock synchronization algorithm is running and once when it is not running.

As *TTP/C* hardware, we use TTPnode and TTPpowernode evaluation boards. At each node, we measure the time differences between the expected and actual arrival of communication messages from other nodes in terms of local clock readings. The *TTP/C* controller that is used on the evaluation boards supports the measurement of these differences. We assume that errors in measurements due to hardware clock drift rates are negligible. We know from Section 2.3 that these differences represent the differences of the clock values at different nodes. The maximum of these time differences is, thus, a good estimate for the precision. The (oscillator) frequencies of the measurement clocks of the time difference are 20 *MHz*. Therefore, the time difference values can only be obtained with a granularity of 50 ns. The measurement unit delivers the value x for time differences that fall into the interval $[\ x \cdot 50\ ns$, $(x+1) \cdot 50\ ns\ ($, for example, the value 2 represents the interval 100 to 150 ns.

Expected Outcome and Hypotheses

As analyzed in the Section 6.4, the maximum influence of the external clock synchronization algorithm on the precision of a cluster is 0.03 ns for typical cluster configuration, such as the experimental setup is. This value is much smaller than can be measured with the available measuring equipment and also by a factor of 10^4 smaller than the precision, which is typically 250 ns to 1 μs. Consequently, we expect the precision values to be the same for the configuration where the algorithm is running and for the configuration where it is not running.

Experiment Results

Figure 7.13 shows a frequency diagram and Table 7.1 shows the frequencies of the measured values at each node (in percent) over a period of 10.24 seconds, which equals 10240 TDMA rounds.

The results show that the precision of an externally synchronized cluster is approximately 250 ns, because the maximum measured time difference is 250 ns. In additional measurements with a duration of 12 hours, where we only recorded the maximum time difference (and not all time difference values), the maximum time difference is also 250 ns.

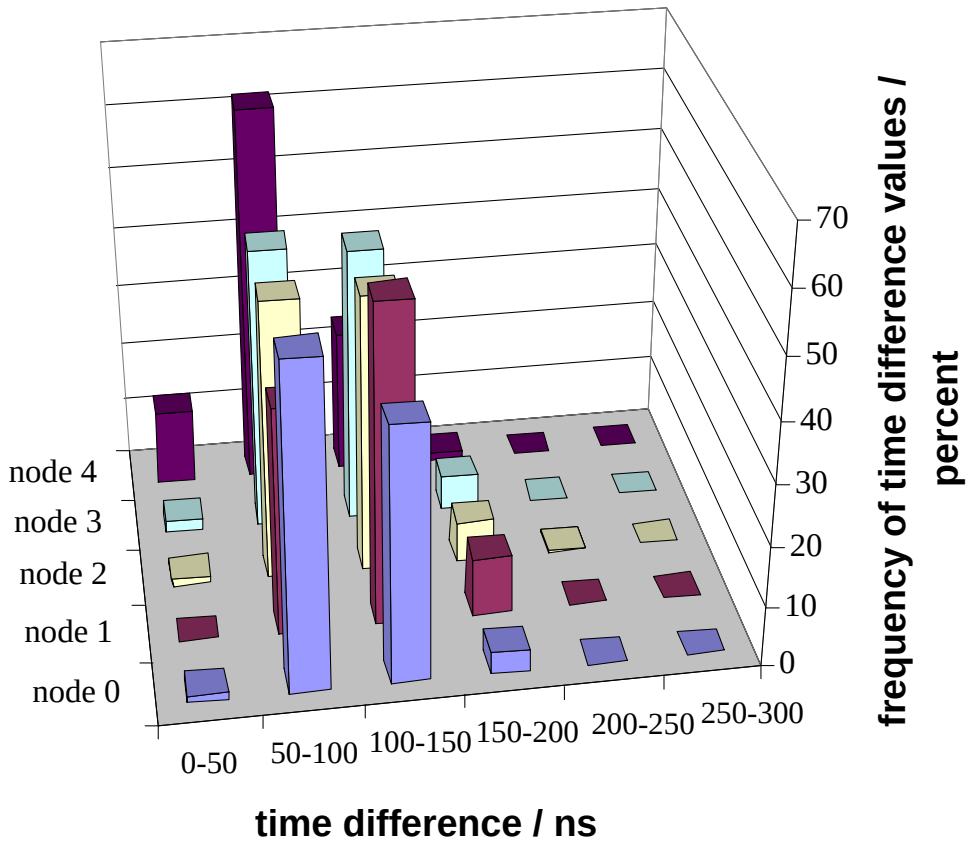


Figure 7.13: Frequency of time difference values (external clock synchronization running)

value (in 50 ns steps)	**node 0** (%)	**node 1** (%)	**node 2** (%)	**node 3** (%)	**node 4** (%)
0 (0-50 ns)	1.22	0.17	1.26	2.01	12.54
1 (50-100 ns)	53.17	37.37	46.26	46.66	62.49
2 (100-150 ns)	41.90	52.60	45.77	45.53	23.97
3 (150-200 ns)	3.69	9.56	6.57	5.67	1.00
4 (200-250 ns)	0.01	0.30	0.14	0.13	0.00
5 (250-300 ns)	0.00	0.00	0.00	0.00	0.00

Table 7.1: Frequency of values representing the difference between expected and actual arrival time of messages (in percent) (external clock synchronization running)

Figure 7.14 and Table 7.2 present the measurements of the time differences over a period of 10.240 seconds (10240 TDMA rounds) without externally syn-

chronizing the cluster. Due to the results, the estimate of the precision of the cluster is 250 ns. Here again, the long-term test (12 hours) has shown the same maximum time difference value.

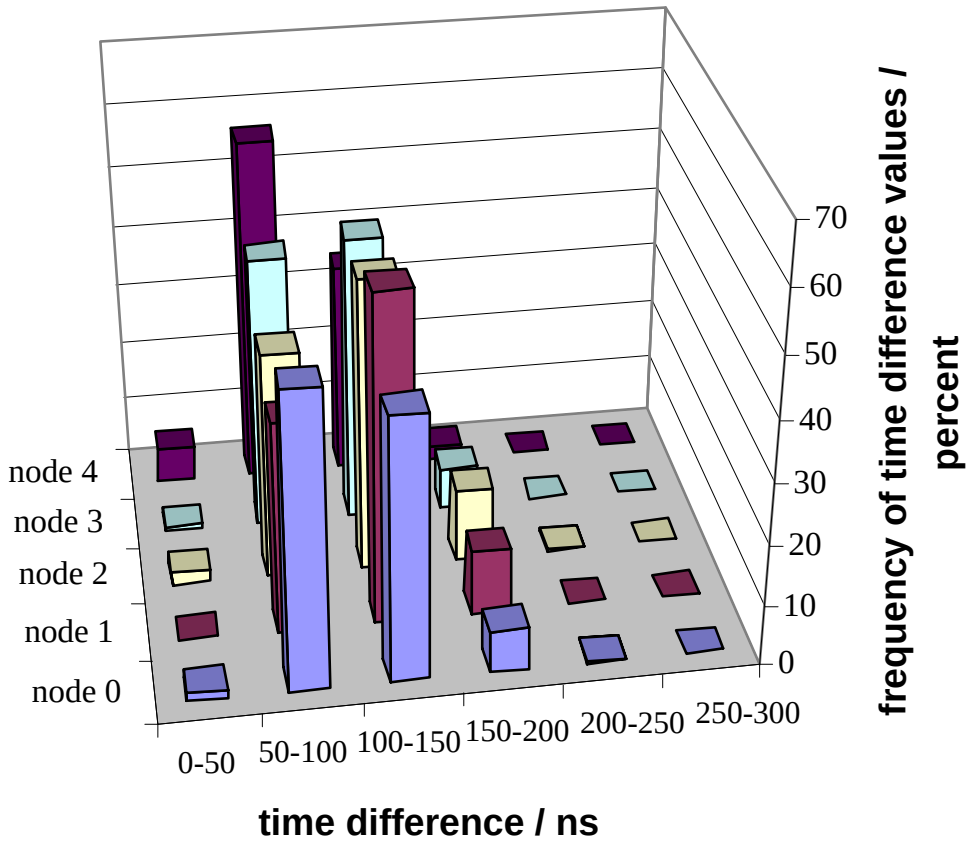


Figure 7.14: Frequency of time difference values (external clock synchronization *not* running)

value (in 50 ns steps)	**node 0** (%)	**node 1** (%)	**node 2** (%)	**node 3** (%)	**node 4** (%)
0 (0-50 ns)	1.46	0.30	2.30	0.96	5.39
1 (50-100 ns)	50.44	36.36	40.00	47.22	55.41
2 (100-150 ns)	41.41	53.16	45.57	44.99	37.01
3 (150-200 ns)	6.61	9.88	11.92	6.62	2.19
4 (200-250 ns)	0.08	0.29	0.21	0.21	0.00
5 (250-300 ns)	0.00	0.00	0.00	0.00	0.00

Table 7.2: Frequency of values representing the difference between expected and actual arrival time of messages (in percent) (external clock synchronization *not* running)

Discussion of Results

We know that the measuring equipment is too imprecise to directly measure the remaining influence calculated in Section 2.3. However, when looking at the different distributions of time difference values when the external clock synchronization algorithm is running and when not, one might be able to observe a small right shift of the distribution, which equals an increased precision value, when the external clock synchronization algorithm is running.

Table 7.1 and 7.2 indicate an influence of digitization errors of the *TTP/C* hardware on the measured time difference values. This can be seen from the different percentages values when looking at the same positions at the two tables. These percentages are off by as much as 13 % (compare values of node 4, value 2). Thus, from a difference in the distribution of the time difference values, one cannot observe an influence (as calculated in Section 2.3) of the external clock synchronization on the precision that is much smaller than the influence of the digitalization errors (even if the measuring equipment would be more precise than the existing one). We could also not measure the remaining influence, because again the digitalization errors of the *TTP/C* hardware are larger than the calculated estimate of Section 2.3. The long-term measurements have lead to an estimate of the precision of the cluster of 250 ns irrespectively of whether the external clock synchronization is running or not. Since there is a difference in different relational clock time values provided at the same time (which is in the order of the precision of a cluster), but we could not measure a difference in the precision, we conclude that external clock synchronization does not interfere the precision of the internal clock synchronization.

7.9 Summary

In this chapter, we presented the experiments that validate the properties of the multi-cluster clock synchronization. The chapter started with a brief description of the experimental setup. The first presented measurements affirm the assumptions of a constant systematic clock error and a symmetrically distributed stochastic one. Based on these results, we investigated the achieved accuracy in a long-term test, examined the elimination of the systematic part of the drift rate and different algorithm-specific parameters, such as the history length. The chapter ends with measurements concerning the influence of the algorithm on the precision of a cluster. The measurements showed that synchronization between multi-cluster systems can be achieved with an accuracy that is better than 1 μs and that non-interference is achieved.

Chapter 8

Conclusion

Nihil est enim simul et inventum et perfectum.

CICERO, *Brutus 71*

The main contribution of the thesis is the development of a clock synchronization algorithm for multi-cluster systems that is especially suited for embedded distributed control systems. Multi-cluster clock synchronization is achieved by externally synchronizing the global time of individual clusters to relational clock times and assuming that global cluster times are not mutually dependent on each other. This work is based on known clock synchronization principles. In contrast to known external clock synchronization algorithms, relational clock times (which are the reference of synchronization) for the external clock synchronization algorithm can have a drift rate different from 0. Thus, relational clock times can either be the global times of other clusters or clocks synchronized to a time standard, such as GPS time.

This multi-cluster clock synchronization strategy is well suited for the needs of embedded distributed control systems, because it addresses the following requirements of embedded distributed control systems:

Tight Synchronization. We define tight synchronization of multi-cluster systems as clock synchronization where the quality of synchronization between clocks of nodes of the different clusters is in the same order of magnitude as the quality of synchronization of clocks of nodes of the same cluster. The presented algorithm achieves tight synchronization of

multi-cluster systems by eliminating systematic errors of clocks and time bases. Due to the design of the algorithm, only stochastic errors of clocks influence the accuracy of a cluster. Since systematic errors of clocks are an order of magnitude larger than stochastic errors in typical distributed control systems, the algorithm achieves a significantly higher quality of synchronization than known algorithms.

Composability. When systems are composed of several systems, composability is a desired property. The multi-cluster clock synchronization provides composability of the time bases, that is, it guarantees that synchronization of the global times of several clusters without changing the precision of the individual clusters despite of arbitrary node faults. We call this property of non-influencing the precision *non-interference*. Non-interference of the algorithm is achieved by providing all nodes with a consistent view of relational clock time values (used as reference) and by exploiting the common notion of time at cluster level.

Dependability. Embedded control systems have increased dependability requirements due to greater autonomy needs compared to desktop computers, which follows from design of embedded systems for stand-alone operation and the expectation of longevity. The presented algorithm can tolerate arbitrarily faulty relational clock times that are used as reference. This is achieved by replication. Furthermore, an analysis of the algorithm has proven that the algorithm is also self-stabilizing. Consequently, synchronization can be maintained despite of arbitrary transient faults.

Low computational complexity. As embedded systems have only low computational power, achieving tight synchronization of cluster time bases using known techniques such as regression analysis of drift offset does not work due to the required computational complexity of regression analysis. The presented algorithm achieves a good approximation of the systematic part of the drift rate of the global time by gradually approximating the systematic part using the average of the drift offset. This approach is very efficient compared to the computation of the regression analysis and needs only one additional measurement round to stabilize.

The presented algorithm has been implemented on top of the Time-Triggered Architecture on two different platforms. Measurements and analysis of the implementation show that the drift rate of the global time of clusters have systematic errors that are approximately ten times larger than stochastic errors. The presented implementation validated that the systematic errors can be eliminated and that tight synchronization can be achieved with low

computational overhead. Measurements show that synchronization between multi-cluster systems can be achieved with an accuracy (measured between the times of the cluster nodes), which is better than 1 μs. If a cluster has access to an accurate external time source that provides exact timing information each full second (such as commercially available GPS receivers), the accuracy of the time at the cluster nodes is better than 1.5 μs.

Furthermore, an analytical model for an estimate of the accuracy in worst-case scenarios has been developed, in which we assume that the errors due to different implementation-specific parameters accumulate. We present these estimates. Briefly described, this accuracy estimate is between 1 and 7 μs for optimally configured clusters. The worst-case estimate is larger than the measured values due to the assumption of accumulating errors originating in different parameters. Errors do not accumulate under normal conditions and in typical specifications. However, the estimate is useful for the analysis of achievable accuracy for real-time systems used in safety-critical and/or dependable systems.

Outlook

The current approach builds on the premise that timing information is only allowed to be transported non-cyclically[1] in multi-cluster systems in order to reach a stable system-wide synchronization. For the presented algorithm, this limitation can be waived if the systematic part of the drift rate does not have to be eliminated. Then, the algorithm will synchronize multi-cluster systems. Yet, a more general approach to extending multi-cluster clock synchronization allowing cyclical transportation of timing information and, thus, feed-back loops is currently subject of another PhD thesis at our institute.

The algorithm has been implemented for validation purposes using the Time-Triggered Architecture. In order to employ the presented concepts in "real-world" applications, implementations on different hardware platforms would be advantageous and another proof-of-concept. The fault tolerance concepts of this algorithm have been analyzed. In addition, fault-injection experiments aiming at testing the fault tolerance concepts could provide additional confidence for deployment of the presented algorithm in highly safety-critical systems.

[1] That is, if cluster A uses the global time of cluster B as reference, cluster B is not allowed to use the global time of cluster A, neither directly nor indirectly.

Bibliography

[AK98a] A. Arora and S.S. Kulkarni. Component based design of multitolerance. *IEEE Transactions on Software Engineering*, 24(1):63–78, 1998.

[AK98b] A. Arora and S.S. Kulkarni. Detectors and correctors: A theory of fault-tolerance components. In *Proceedings of the* 18^{th} *International Conference on Distributed Computing Systems*, pages 436–443, Amsterdam, The Netherlands, 1998. IEEE Computer Society.

[Ale64] C. Alexander. *Notes on the Synthesis of Form.* Harvard University Press, Cambridge, MA, USA, and London, England, United Kingdom, 1964.

[ANS89] The Advanced Network Systems Architecture (ANSA) Reference Model. Technical report, Architecture Projects Management Ltd., Castle Hill, Cambridge, England, 1989.

[AP95] T.K. Apostolopoulos and K.C. Pramataris. A client-server architecture for distributed problem solving. In *Proceedings of IEEE International Conference on Information Engineering*, pages 513–517, Singapore, 1995.

[AP97] E. Anceaume and I. Puaut. A taxonomy of clock synchronization algorithms. Research Report 1103, Institut National de Recherche en Informatique et Systèmes Aléatoires (IRISA), Rennes, France, July 1997.

[AP98] E. Anceaume and I. Puaut. Performance evaluation of clock synchronization algorithms. Research Report 3526, Institut National de Recherche en Informatique et Systèmes Aléatoires (IRISA), Rennes, France, October 1998.

[ARI93] *ARINC Specification 659: Backplane Data Bus*. Aeronautical Radio, Inc., Annapolis, MD, USA, December 1993. Prepared by the Airlines Electronic Committee.

[Avi78] A. Avizienis. Fault tolerance, the survival attribute of digital systems. *Proceedings of IEEE*, 66(10):1109–1125, October 1978.

[AW98] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. McGraw-Hill Publishing Company, Berkshire, England, 1998.

[Bau01] G. Bauer. *Transparent Fault Tolerance in a Time-Triggered Architecture*. Doctoral thesis, Institut für Technische Informatik, Technische Universität Wien, Treitlstr. 1-3/3/182-1, Vienna, Austria, May 2001.

[BCE+01] G. Borriello, R.P. Colwell, D.L. Estrin, J. Fiddler, M. Horowitz, W.J. Kaiser, N.G. Leveson, B.H. Liskov, P. Lucas, D.P. Maher, P. Mankiewich, R. Taylor, and J. Waldo. *Embedded, Everywhere: A Research Agenda for Networked Systems of Embedded Computers*. National Academy Press, 2001. Publication of the Committee on Networked Systems of Embedded Computers, Computer Science and Telecommunications Board, National Research Council.

[BD87] O. Babaoglue and R. Drummond. (Almost) no cost clock synchronization. In *Proceedings of the 7^{th} International Symposium on Fault-Tolerant Computing*, pages 42–47, Pittsburgh, PE, USA, July 1987. IEEE Computer Society Press.

[Ben90] M. Ben-Ari. *Principles of Concurrent and Distributed Programming*. Prentice Hall international series in computer science, C.A.R. Hoare, series editor. Prentice Hall International (UK) Ltd, Hertfordshire, United Kingdom, 1990.

[BK00] G. Bauer and H. Kopetz. Transparent redundancy in the Time-Triggered Architecture. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 5–13. IEEE Computer Society Press, June 2000.

[BKP01] G. Bauer, H. Kopetz, and P. Puschner. Assumption coverage under different failure modes in the Time-Triggered Architecture. In *Proceedings of the IEEE Conference on Emerging Technologies in Factory Automation*, Antibes Juan les Pins, France, October 2001.

[BKS02] G. Bauer, H. Kopetz, and W. Steiner. The central guardian approach to enforce fault isolation in a time-triggered system. Research Report 20/2002, Institut für Technische Informatik, Technische Universität Wien, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2002.

[BP00] G. Bauer and M. Paulitsch. An investigation of membership and clique avoidance in TTP/C. In *Proceedings of the* 19^{th} *IEEE Symposium on Reliable Distributed Systems, 2000*, pages 118–124, Nuremberg, Germany, October 2000. IEEE Press.

[BS01] G. Bauer and W. Steiner. Smart bus guardian design guidelines. Research Report 20/2001, Institut für Technische Informatik, Technische Universität Wien, Treitlstr. 1-3/3/182-1, Vienna, Austria, 2001.

[Car82] W.C. Carter. A time for reflection. In *Proceedings of the* 12^{th} *International Symposium on Fault-Tolerant Computing*, page 41, Santa Monica, CA, USA, June 1982. IEEE Computer Society Press.

[CDK94] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concept and Design.* Addison-Wesley Publishing Ltd., 2^{nd} edition, 1994.

[CF95] F. Cristian and C. Fetzer. Fault-tolerant external clock synchronization. In *Proceedings of the* 15^{th} *International Conference on Distributed Computing Systems*, pages 70–77, Los Alamitos, CA, USA, May 30–June 2 1995. IEEE.

[CF98] F. Cristian and C. Fetzer. The timed asynchronous distributed system model. In *Proceedings of the* 28^{th} *International Symposium on Fault-Tolerant Computing*, Munich, Germany, June 1998.

[CF99] F. Cristian and C. Fetzer. The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):642–657, June 1999.

[CHTC96] T. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost. On the impossibility of group membership. In *Proceedings of the* 15^{th} *ACM Symposium on Principles of Distributed Computing*, pages 322–330, Philadelphia, PA, USA, May 1996.

[CL94] G. Ciardo and C. Lindemann. Comments on "analysis of self-stabilizing clock synchronization by means of stochastic petri

nets". *IEEE Transactions on Computers*, 43(12):1453–1456, 1994.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. The MIT Press / McCraw-Hill Book Company, Cambridge, MA, USA / New York, NY, USA, 1^{st} edition, 1990.

[Cou85] P.-J. Courtois. On time and space decomposition of complex structures. *Communications of the ACM*, 28(6):590–603, 1985.

[Cri89] F. Cristian. Probabilistic clock synchronization. *Distributed Computing*, 3:146–158, 1989.

[Cri91] F. Cristian. Understanding fault-tolerant distributed systems. *Communications of the ACM*, 34(2):56–78, 1991.

[CS68] W.C. Carter and P.R. Schneider. Design of dynamically checked computers. In *IFIP'68 Congress*, pages 878–883, Amsterdam, The Netherlands, 1968.

[Dan97] P.H. Dana. Global Posinitioning System (GPS) time dissemination for real-time applications. *Real-Time Systems*, 12(1):9–40, January 1997.

[DDS87] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM (JACM)*, 34(1):77–97, 1987.

[Dij68] E.W. Dijkstra. Cooperating sequential processes. In F. Genuys, editor, *Programming Languages*, pages 43–112, NY, USA, 1968. Academic Press.

[Dij74] E.W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[Dij82] E.W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Selected Writings on Computing: A Personal Perspective*, pages 41–46, 1982. Springer Verlag, Berlin. Originally published in 1973.

[DLS88] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[dM95] J. de Meer. The ISO Reference Model for Open Distributed Processing. *Computer Networks and ISDN Systems*, 27(8):1211–1214, July 1995.

[Dol97] S. Dolev. Possible and impossible self-stabilizing digital clock synchronization in general graphs. *Real-Time Systems*, 12(1):95–107, January 1997.

[DW95] S. Dolev and J.L. Welch. Self-stabilizing clock synchronization with byzantine faults. In *Proceedings of the* 14^{th} *ACM Symposium on Principles of Distributed Computing*, page 256. ACM Press, 1995.

[FC95] C. Fetzer and F. Cristian. An optimal internal clock synchronization algorithm. In *Proceedings of the* 10^{th} *Conference on Computer Assurance*, pages 187–196, Gaithersburg, MD, USA, June 1995. IEEE.

[FC97] C. Fetzer and F. Cristian. Integrating external and internal clock synchronization. *Real-Time Systems*, 12:123–171, March 1997.

[Fis90] M. Fischer. A theoreticians view of fault tolerant distributed computing. In B. Simons and A. Spector, editors, *Fault-Tolerant Distributed Computing*, volume 448 of *Lecture Notes in Computer Sciences*, pages 1–9. Springer-Verlag, Berlin, Germany, 1990.

[FLP85] M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[Gho93] S. Ghosh. An alternative solution to a problem on self-stabilization. *ACM Transactions on Programming Languages and Systems*, 15(4):735–742, 1993.

[GM91] M.G. Gouda and N.J. Multari. Stabilizing communication protocols. *IEEE Transactions on Computers*, 40(4):448–458, April 1991.

[HD93] K. Hoyme and K. Driscoll. Safebus (tm). *IEEE Aerospace and Electronics Systems Magazine*, 8(3):34–39, March 1993.

[HLD88] R.E. Harper, J.H. Lala, and J.J. Deyst. Fault tolerant parallel processor architecture overview. In *Proceedings of the* 18^{th} *International Symposium on Fault-Tolerant Computing*, pages 252–257. IEEE Computer Society Press, June 1988.

[HLMR90] J.J. Horning, H.C. Lauer, P.M. Melliar-Smith, and B. Randell. A program structure for error detection and recovery. In E. Gelenbe

and C. Kaiser, editors, *Proceedings of an International Symposium held at Rocquencourt*, volume 16 of *Lecture Notes in Computer Sciences*, pages 171–187. Springer-Verlag, Berlin, Germany, 1990.

[HS97] D. Höchtl and U. Schmid. Long-term evaluation of GPS timing receiver failures. In *Proceedings of the* 29^{th} *Precise Time and Time Interval Systems and Applications Meeting*, Long Beach, USA, December 1997.

[HSSD84] J.Y. Halpern, B. Simons, R. Strong, and D. Dolev. Fault-tolerant clock synchronization. In *Proceedings of the* 3^{rd} *ACM Symposium on Principles of Distributed Computing*, pages 89–102, 1984.

[HT93] V. Hadzilacos and S. Toueg. *Distributed Systems.* Addison-Wesley, 1993.

[HT94] V. Hadzilacos and S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR94-1425, Department of Computer Science, Cornell University, Ithaca, NY, USA, May 1994.

[IEE02] *Draft Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems (V0.19.13).* IEEE Press, New York, NY, USA, May 2002. IEEE Standard No. P1588; Product No. DS5905-TBR.

[Int92] International Standards Organization. Basic Reference Model of Open Distributed Processes, Part 1: Overview and guide to use. Technical Report ISO/IEC JTC1/SC212/WG7 CD 10746-1, International Standards Organization, 1992.

[Jal94] P. Jalote. *Fault Tolerance in Distributed Systems.* Prentice Hall, Inc., Englewood Cliffs, NJ, USA, 1994.

[Jen02] S. Jennings. Hier ticken alle Uhren gleich: IEEE1588 – die Synchronisation über Ethernet. *IEE Automatisierung + Datentechnik*, 47(7):52–55, July 2002.

[JKK+01] C. Jones, M.-O. Killijian, H. Kopetz, E. Marsden, N. Moffat, M. Paulitsch, D. Powell, B. Randell, A. Romanovsky, and R. Stroud. Revised version of DSoS conceptual model. Project Deliverable IC1 for Project "Dependable Systems of Systems" (DSoS) / Research Report 35/2001, Institut für Technische Informatik, Technische Universität Wien, Treitlstr. 1-3/3/182-1, Vienna, Austria, 2001.

[KB02] H. Kopetz and G. Bauer. The Time-Triggered Architecture. *Proceedings of the IEEE*, 90(10), October 2002. Special issue on Modeling and Design of Embedded Software.

[KBJ00] L.M. Kaufman, S. Bhide, and B.W. Johnson. Modeling of common-mode failures in digital embedded systems. In *Proceedings of the Annual Reliability and Maintainability Symposium*, pages 350–357, Los Angeles, CA, USA, January 2000. IEEE, IEEE Press.

[Kha01] M.S. Khan. Political and economic dimensions of Global Navigation Satellite System (GNSS). In *IEEE Proceedings of the Aerospace Conference*, volume 3, pages 3/1271 – 3/1276. IEEE, 2001.

[KHE01] H. Kopetz, M. Holzmann, and W. Elmenreich. A universal smart transducer interface: TTP/A. *International Journal of Computer System, Science & Engineering*, 16(2), March 2001.

[KHK+96] H. Kopetz, R. Hexel, A. Krüger, D. Millinger, and A. Schedl. A synchronization strategy for a TTP/C controller. In *Society of Automotive Engineers International Congress and Exposition*, Detroit, MI, USA, February 1996. SAE International.

[Kim00] K.H. Kim. Issues insufficiently resolved in century 20 in the fault-tolerant distributed computing field. In *IEEE Symposium on Reliable and Distributed Systems*, pages 106–115, Nuremberg, Germany, October 2000. IEEE Computer Society Press.

[KKMS95] H. Kopetz, A. Krüger, D. Millinger, and A. Schedl. A synchronization strategy for a time-triggered multicluster real-time system. In *Symposium on Reliable Distributed Systems*, pages 154–161, Los Alamitos, CA, USA, September 1995. IEEE Computer Society Press.

[KO87] H. Kopetz and W. Ochsenreiter. Clock synchronization in distributed real time systems. *IEEE Transactions on Computers*, 36(8), August 1987.

[KO02] H. Kopetz and R. Obermaisser. Temporal composability. *IEE Computing & Control Engineering Journal*, 13(8), August 2002.

[Kop97] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.

[Kop98] H. Kopetz et al. Specification of the TTP/C protocol – chip version 0.1. Technical Report 27/1998, Institut für Technische Informatik, Technische Universität Wien, Treitlstr. 1-3/3/182-1, Vienna, Austria, December 1998.

[Kop99] H. Kopetz. *TTP/C Protocol*. TTTech, Vienna, Austria, July 1999. Available at `http://www.ttpforum.org/`.

[Kop00a] H. Kopetz. Composability in the Time-Triggered Architecture. In *Society of Automotive Engineers World Congress*, Detroit, MI, USA, March 2000. Society of Automotive Engineers, International. Paper No. 2000-01-1382.

[Kop00b] H. Kopetz. Software engineering for real-time: A roadmap. In A. Finkelstein, editor, *Proceedings of the* 22^{nd} *International Conference on the Future of Software Engineering*, pages 201–211, Limerick, Ireland, June 2000. ACM Press.

[Kop01a] H. Kopetz. Note on the drift of the global time in a TTP cluster. Research Report 1/2001, Institut für Technische Informatik, Technische Universität Wien, Treitlstr. 1-3/3/182-1, Vienna, Austria, January 2001.

[Kop01b] H. Kopetz. The temporal specification of interfaces in distributed real-time systems. In C.M. Kirsch T.A. Henzinger, editor, *Embedded Software, Proceedings of* 1^{st} *International Workshop on Embedded Software*, volume 2211 of *Lecture Notes in Computer Science*, pages 223–236. Springer-Verlag Berlin Heidelberg, Germany, Lake Tahoe, CA, USA, October 2001.

[Kop02a] H. Kopetz. Fault containment und error detection in the Time-Triggered Architecture. Research Report 39/2002, Institut für Technische Informatik, Technische Universität Wien, Treitlstr. 1-3/3/182-1, Vienna, Austria, 2002.

[Kop02b] H. Kopetz. Time-triggered real-time computing. In *Proceedings of the IFAC World Congress on Automatic Control*, Barcelona, Spain, July 2002. IFAC Press.

[KR87] Shin K.G. and Ramanathan R. Clock synchronization of large multiprocessor system in the presence of malicious faults. *IEEE Transactions on Computers*, 36(1):2–12, 1987.

[KR93] H. Kopetz and J. Reisinger. The non-blocking write protocol NBW: A solution to a real-time synchronization problem. In *Proceedings of the* 14^{th} *Real-Time Systems Symposium*, pages 131–137, Raleigh-Durham, NC, USA, December 1993.

[KS02] H. Kopetz and N. Suri. Compositional design of real-time systems: A conceptual basis for the specification of linking interfaces. Research Report 8/2002, Institut für Technische Informatik, Technische Universität Wien, Treitlstr. 1-3/3/182-1, Vienna, Austria, 2002.

[Kul99] S.S. Kulkarni. *Component-Based Design of Fault Tolerance*. PhD thesis, The Ohio State University, Columbus, OH, USA, June 1999.

[LA90] P.A. Lee and T. Anderson. *Fault Tolerance*. Springer Verlag, Vienna, Austria, 2^{nd} edition, 1990. Dependable Computing and Fault-Tolerant Systems. Vol. 3. A. Avižienis, H. Kopetz, and J.-C. Laprie (editors), IFIP WG 10.4 Dependable Computing and Fault Tolerance.

[LAK99] W. Lewandowski, J. Azoubib, and W.J. Klepczynski. GPS: Primary tool for time transfer. *Proceedings of the IEEE*, 87(1):163–172, January 1999.

[Lap92] J.-C. Laprie. *Dependability: Basic Concepts and Terminology*. Springer Verlag, Vienna, Austria, 1992. Dependable Computing and Fault-Tolerant Systems. Vol. 5. A. Avižienis, H. Kopetz, and J.-C. Laprie (editors), IFIP WG 10.4 Dependable Computing and Fault Tolerance.

[Lap95] J.-C. Laprie. Dependability – its attributes, impairments, and means. In B. Randell, J.-C. Laprie, H. Kopetz, and B. Littlewood, editors, *Predictably Dependable Computing Systems*, pages 3–24, Heidelberg, Germany, 1995. Springer Verlag.

[Leb98] M. Lebedev. GLONASS as instrument for precise UTC transfer. In *Proceedings of the* 12^{th} *European Frequency and Time Forum*, Warsaw, Poland, March 1998.

[Lee99] E. A. Lee. Embedded software – an agenda for research. ERL Technical Report UCB/ERL No. M99/63 M99/63, University of California, Berkeley, CA, USA, December 1999.

[LFA90] J. Long, W.K. Fuchs, and J.A. Abraham. Forward recovery using checkpointing in parallel systems. In *Proceedings of the International Conference on Parallel Processing*, pages 272–275, August 1990.

[Lic97] R. Lichtenecker. Terrestrial time signal dissemination. *Real-Time Systems*, 12(1):41–61, January 1997.

[Lis91] B. Liskov. Practical use of synchronized clocks in distributed systems. In *Proceedings of* 10^{th} *ACM Symposium on the Principles of Distributed Computing*, pages 1–9. ACM Press, 1991.

[LL84a] J. Lundelius and N. Lynch. A new fault-tolerant algorithm for clock synchronization. In *Proceedings of the* 3^{rd} *annual ACM symposium on Principles of Distributed Computing*, pages 75–88. ACM, 1984.

[LL84b] J. Lundelius and N. Lynch. An upper and lower bound for clock synchronization. *Information and Control*, 62:190–204, 1984.

[LM84] L. Lamport and P.M. Melliar-Smith. Byzantine clock synchronization. In *Proceedings of the* 3^{rd} *ACM Symposium on Principles of Distributed Computing*, pages 68–74, 1984.

[LM85] L. Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM*, 32(1):52–78, January 1985.

[LSP82] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[LZM90] M. Lu, D. Zhang, and T. Murata. Analysis of self-stabilizing clock synchronization by means of stochastic petri nets. *IEEE Transactions on Computers*, 39(5):597–604, 1990.

[Mar84] K.A. Marzullo. *Maintaining the Time in a Distributed System: An Example of a Loosely Coupled Distributed Service*. PhD thesis, Department of Electrical Engineering, Stanford University, Stanford, CA, USA, February 1984.

[MHB+01] R. Mores, G. Hay, R. Belschner, J. Berwanger, C. Ebner, S. Fluhrer, E. Fuchs, B. Hedenetz, W. Kuffner, A. Krüger, P. Lohrmann, D. Millinger, M. Peller, J. Ruh, A. Schedl, and

M. Sprachmann. FlexRay – the communication system for advanced automotive control systems. In *Society of Automotive Engineers World Congress*, Detroit, MI, USA, March 2001. SAE International. Document No 2001-01-0676.

[Mil91] D.L. Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, October 1991.

[Min00] P.S. Miner. Analysis of the SPIDER fault-tolerance protocols. In *LFM 2000:* 5^{th} *NASA Langley Formal Methods Workshop*, NASA Langley Research Center, Hampton, VA, June 2000. C. Michael Holloway, editor. Slides available at `http://shemesh.larc.nasa.gov/fm/Lfm2000/Presentations/lfm2000-spider/`.

[MO83] K. Marzullo and S. Owicki. Maintaining the time in a distributed system. In *Proceedings of the* 2^{nd} *ACM Symposium on Principles of Distributed Computing*, pages 295–305, 1983.

[MS85] S.R. Mahaney and F.B. Schneider. Inexact agreement: accuracy, precision, and graceful degradation. In *Proceedings of the* 4^{th} *ACM Symposium on Principles of Distributed Computing*, pages 237–249. ACM Press, 1985.

[MT89] M.D. Mesarovic and Y. Takahara. *Abstract Systems Theory*, volume 116 of *Lecture Notes in Control and Information Sciences*. Springer-Verlag Berlin, Heidelberg, Germany, 1989.

[Mye88] B. Myer. *Object-Oriented Software Construction*. Prentice Hall, New York, NY, USA, 1988.

[OMG02] Smart transducers interface. Specification ptc/2002-05-01, Object Management Group, May 2002. Available at `http://www.omg.org/`.

[Pas02] A. Pasetti. *Software Frameworks and Embedded Control Systems*, volume 2231 of *Lecture Notes in Computer Science*. Springer-Verlag Berling Heidelberg, Germany, ETH-Zentrum, Zurich, Switzerland, 2002.

[Pat94] B. Patt. *A Theory of Clock Synchronization*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institut of Technology, Cambridge, MA, USA, October 1994.

[PB95] M. Pfluegl and D. Blough. A new and improved algorithm for fault-tolerant clock synchronization. *Journal of Parallel and Distributed Computing*, 27:1–14, 1995.

[PB00] M. Paulitsch and G. Bauer. External clock synchronization in the Time-Triggered Architecture (TTA). Research Report 3/2000, Institut für Technische Informatik, Technische Universität Wien, Treitlstr. 1-3/3/182-1, Vienna, Austria, April 2000.

[Pol94] S. Poledna. Replica determinism in distributed real-time systems: A brief survey. *Real-Time Systems*, 6:289–316, 1994.

[Pow92] D. Powell. Failure mode assumptions and assumption coverage. In *Proceedings of the* 22^{th} *International Symposium on Fault-Tolerant Computing*, pages 386–395, Boston, MA, USA, July 1992. IEEE Computer Society Press. Revised version in predictably dependable computing systems, Eds. B. Randell, J.-C. Laprie, H. Kopetz, B. Littlewood, Springer Verlag, NISBN 3-540-59334-9, 1995, pp.123–140.

[Pow94] D. Powell. Distributed Fault Tolerance – Lessons Learnt from Delta-4. In M. Banatre and P.A. Lee, editors, *Hardware and Software Architectures for Fault Tolerance: Experiences and Perspectives*, volume 774 of *Lecture Notes in Computer Sciences*, pages 199–217. Springer-Verlag, Berlin, Germany, 1994. Also published as LAAS research report No. 93192. Workshop on Fault-Tolerant Architectures, Mont Saint Michel, France, June, 1993.

[PS01] D. Powell and R. Stroud (editors). Conceptual model and architecture. Project deliverable D2 for project MAFTIA (Malicious- and Accidental-Fault Tolerance for Internet Applications) Technical Report CS-TR-749, University of Newcastle upon Tyne, Technical Report DI/FCUL TR-01-10, Universidade de Lisboa, LAAS-CNRS Report No. 01426, Research Report RZ 3377, IBM Research, Zurich Research Laboratory, LAAS-CNRS / University of Newcastle upon Tyne, Toulouse, France / Newcaste, England, United Kingdom, November 2001.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[PT94] M. Papatriantafilou and P. Tsigas. Self-stabilizing wait-free clock synchronization. In *Proceedings of the* 4^{th} *Scandinavian Work-*

shop on Algorithm Theory, volume 824 of *Lecture Notes in Computer Science*, pages 267–277. Springer-Verlag Berlin Heidelberg, Germany, July 1994.

[PV94] D.K. Pradhan and N.H. Vaidya. Roll-forward checkpointing scheme: A novel fault-tolerant architecture. *IEEE Transactions on Computers*, 43(10):1163–1174, 1994.

[Rec91] E. Rechtin. *System Architecting: Creating and Building Complex Systems*. Prentice-Hall, Inc., Englewood Cliffs, NJ, USA, 1991.

[RSB90] P. Ramanathan, K.G. Shin, and R.W. Butler. Fault-tolerant clock synchronization in distributed systems. *IEEE Computer*, 23(10):33–42, October 1990.

[Rus02] J. Rushby. Self-stabilization of the TTA group membership algorithm. CSL Technical Report 24b, Computer Science Laboratory, SRI International, Menlo Park, CA, USA, February 2002.

[RvH89] J. Rushby and F. von Henke. Formal verification of the interactive convergence clock synchronization algorithm. Technical Report CSL-89-3R, Computer Science Laboratory, SRI International, CA, USA, February 1989.

[Sat88] M. Satyanarayanan. On the influence of scale in a distributed system. In *Proceedings of the* 10^{th} *International Conference on Software Engineering*, pages 10–18, Singapore, 1988. IEEE Press.

[Sat93] M. Satyanarayanan. Distributed file system. In S. Mullender, editor, *Distributed Systems*, pages 353–383. Addison-Wesley Longman, Inc., ACM Press, 1993.

[SB02] W. Steiner and G. Bauer. Using a central guardian for a fault-tolerant system startup of TTP/C. Research Report 16/2002, Institut für Technische Informatik, Technische Universität Wien, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2002.

[Sch87] F.B. Schneider. Understanding protocols for byzantine clock synchronization. Research Report 87-859, Department of Computer Science, Cornell University, Ithaca, NY, USA, August 1987.

[Sch88] W. Schwabl. *Der Einfluss zufälliger und systematischer Fehler auf die Uhrensynchronisation in verteilten Echtzeitsystemen.* Doctoral thesis, Institut für Technische Informatik, Technische Universität Wien, Treitlstr. 1-3/3/182-1, Vienna, Austria, October 1988.

[Sch93a] M. Schneider. Self-stabilization. *ACM Computing Surveys*, 25(1):45–67, 1993.

[Sch93b] M.D. Schroeder. A state-of-the-art distributed system: Computing with BOB. In S. Mullender, editor, *Distributed Systems*, pages 1–16. Addison-Wesley Longman, Inc., ACM Press, 1993.

[Sch96] A. Schedl. *Design and Simulation of Clock Synchronization in Distributed Systems*. Doctoral thesis, Institut für Technische Informatik, Technische Universität Wien, Treitlstr. 1-3/3/182-1, Vienna, Austria, April 1996.

[Sch00] U. Schmid. Orthogonal accuracy clock synchronization. *Chicago Journal of Technical Computer Science*, 2000(3):3–77, August 2000.

[SJ82] D.P. Siewiorek and D. Johnson. A design methodology for high reliability systems: The Intel 432. In D.P. Siewiorek and R.S. Swarz, editors, *The Theory and Practice of Reliable System Design*, pages 621–636. Digital Press, 1982.

[SP02] W. Steiner and M. Paulitsch. The transition from asynchronous to synchronous system operation: An approach for distributed fault-tolerant systems. In *Proceedings of the* 22^{nd} *International Conference on Distributed Computing Systems*, Vienna, Austria, July 2002. IEEE Press.

[SR88] J.A. Stankovic and K. Ramamritham. *Hard real-time systems: Tutorial*. IEEE Computer Society Press, Washington, D.C., USA, 1988.

[SS92] D.P. Siewiorek and R.S. Swarz. *Reliable Computer Systems: Design and Evaluation*. Digital Press, Bedford, MA, USA, 2^{nd} edition, 1992.

[SS97] U. Schmid and K. Schossmaier. Interval-based clock synchronization. *Real-Time Systems*, 12:173–228, March 1997.

[ST87] T.K. Srikanth and S. Toueg. Optimal clock synchronization. *Journal of the ACM*, 34(3):626–645, 1987.

[Sta01] W. Stallings. *Operating Systems*. Prentice-Hall, Inc., Englewood Cliffs, NJ, USA, 4^{th} edition, 2001.

[Ste02a] W. Steiner. Investigating information systems. Research Report 38/2002, Institut für Technische Informatik, Technische Universität Wien, Treitlstr. 1-3/3/182-1, Vienna, Austria, 2002.

[Ste02b] W. Steiner. Self-stabilizing in the TTA despite permanent failures. Research Report 21/2002, Institut für Technische Informatik, Technische Universität Wien, Treitlstr. 1-3/3/182-1, Vienna, Austria, 2002.

[Szy99] C. Szyperski. *Component Software: Beyond Object-oriented Programming*. ACM Press, London, Great Britain, 1^{st} edition, 1999.

[TTT01] TTTech Computertechnik AG. Technical documentation of TTPnode and TTPpowernode. Available at `http://www.tttech.com/`, 2001.

[Vit96] Vitruvius. *De architectura libri decem / Zehn Bücher über Architektur*. Primus Verlag, Darmstadt, Germany, 5^{th} edition, originally around 23 B.C. / 1996. Translated and commented by C. Fensterbusch.

[VRC97] P. Veríssimo, L. Rodrigues, and A. Casimiro. CesiumSpray: A precise and accurate global time service for large-scale systems. *Real-Time Systems*, 12(3):243–294, May 1997.

[Weg84] P. Wegner. Capital-intensive software technology. *IEEE Software*, 1(3), 1984.

[WF00] H.F. Wedde and W. Freund. Harmonious internal clock synchronization. In 12^{th} *Euromicro Conference on Real-Time Systems*, pages 175–182, Informatik III, Dortmund University, Dortmund, Germany, June 2000. IEEE Press.

[WWWK97] J. Waldo, G. Wyant, A. Wollrath, and S. Kendall. A Note on Distributed Computing. In J. Vitek and C. Tschudin, editors, *Mobile Object Systems: Towards the Programmable Internet*, volume 1222 of *Lecture Notes in Computer Science*, pages 49–64. Springer Verlag, Heidelberg, Germany, April 1997. Also published as Sun Microsystems Laboratories Technical Report SMLI TR-94-29.

[YC75] S.S. Yau and R.C. Cheung. Design of self-checking software. In 1^{st} *International Conference on Reliable Software*, pages 450–457, Los Angeles, CA, USA, 1975.

Appendix A

Notation

Table A.1 provides an overview of the most important terms that are used in this thesis.

Symbol	Description
$\mathbf{t}$	point in real time
$\mathcal{T}$	set of all real-time values, the granularity of real time is infinitesimal
$\mathcal{RT}$	reference clock time is a granular representation of $\mathcal{T}$; that is, in perfect synchrony with $\mathcal{T}$
$\mathcal{RT}(\mathbf{e})$	timestamp of occurrence of event e using $\mathcal{RT}$
$\mathcal{HT}^{\mathbf{i}}$	hardware clock time of clock i given by physical oscillator
$\mathcal{VT}^{\mathbf{i}}$	virtual clock time i is an abstraction of $\mathcal{HT}^i$ and can be adjusted by adding a correction term $(H^i(t))$ to stay in synchrony with other virtual clock times
$\mathcal{VT}^{\mathbf{i}}(\mathbf{t})$	clock reading of virtual clock i at t
$\mathcal{VT}^{\mathbf{i}}_{\mathbf{r}}$	clock tick r of $\mathcal{VT}^i$
$\mathbf{M}$	number of time master nodes in a cluster; time master nodes have access to relational clock times
$\mathcal{RCT}^{\mathbf{j}}$	relational clock time (also called external time source if synchronized to a time standard); time master node j, $1 \leq j \leq M$, has access to $\mathcal{RCT}^j$
$\mathcal{RCT}^{\mathbf{j}}_{\mathbf{r}}$	clock tick r of $\mathcal{RCT}^j$
$\mathcal{RCT}^{\mathbf{j}}(\mathbf{t})$	clock reading of $\mathcal{RCT}^j$ at t
$\mathbf{\Delta}$	maximum deviation between all $\mathcal{RCT}$ of a cluster; $\Delta = max_{1\leq i,j\leq M,\forall k}(\lvert\mathcal{RT}(\mathcal{RCT}^i_k) - \mathcal{RT}(\mathcal{RCT}^j_k)\rvert)$
$\mathbf{O}^{\mathbf{i,j}}_{\mathbf{k}}$	the offset between $\mathcal{VT}^i$ and $\mathcal{VT}^j$ for tick k; $O^{i,j}_k = \lvert\mathcal{RT}(\mathcal{VT}^i_k) - \mathcal{RT}(\mathcal{VT}^j_k)\rvert)$

Symbol	Description
$\mathbf{\Omega_t^j}$	offset between virtual clock time and relational clock time at t as can be read by time master node j; $\Omega_t^j = \left\lvert \mathcal{VT}^j(t) - \mathcal{RCT}^j(t) \right\rvert$
$\mathbf{N}$	number of synchronizing nodes in a cluster (including time master nodes);
$\mathbf{\Pi}$	precision of a cluster; $\Pi = \max_{1 \leq i,j \leq N, interval\ of\ interest}(O_k^{i,j})$
VT	global cluster time VT is an abstraction of all $\mathcal{VT}^i$, $1 \leq i \leq N$, that are synchronized to each other with precision Π
$\mathbf{R_{measure}^n}$	measurement interval n
$\mathbf{R_{ext.CS}^m}$	external clock synchronization interval m
$\rho^{\mathbf{i}}$	drift rate of clock i $\rho^i = \left\lvert \frac{\mathcal{RT}(\mathcal{VT}_{r+1}^i) - \mathcal{RT}(\mathcal{VT}_r^i)}{n_r} - 1 \right\rvert$, where n_r is the nominal number of reference clock ticks of clock tick r
$\bar{\rho}^{\mathbf{i}}$	maximum drift rate of clock i
$\rho_{\mathbf{stoch}}^{\mathbf{i}}$	stochastic part of drift rate of clock i
$\rho_{\mathbf{sys}}^{\mathbf{i}}$	systematic part of drift rate of clock i
$\tilde{\rho}_{\mathbf{sys}}^{\mathbf{i,d}}$	estimated systematic part of drift rate of clock i in interval d
$\alpha^{\mathcal{VT}^{\mathbf{j}},\mathcal{RCT}^{\mathbf{j}}}$	accuracy of $\mathcal{VT}^j$ with respect to $\mathcal{RCT}^j$;
$\alpha^{\mathcal{VT}^{\mathbf{i}},\mathcal{T}}$	accuracy of $\mathcal{VT}^i$ with respect to $\mathcal{T}$

Table A.1: Overview of notation

Appendix B

Measurement Data

This section presents the measurement data that is discussed in Section 7 and depicted in Figures 7.1, 7.4, 7.5, 7.6, 7.7, 7.8, 7.9, 7.10, 7.11, and 7.12 in tables.

time (s)	**deviation of calibrated cluster** (ns)	**deviation of not calibrated cluster** (ns)
1	190.7	190.7
2	190.7	0.0
3	190.7	0.0
4	-190.7	0.0
5	-190.7	190.7
6	-190.7	0.0
7	0.0	-190.7
8	-190.7	0.0
9	-190.7	-190.7
10	381.4	190.7
11	190.7	-190.7
12	0.0	190.7
13	0.0	-190.7
14	-190.7	0.0
15	0.0	190.7
16	190.7	-190.7
17	0.0	190.7
18	190.7	190.7
19	572.1	381.4

time	deviation of calibrated cluster	deviation of not calibrated cluster
20	381.4	190.7
21	190.7	0.0
22	381.4	190.7
23	0.0	-190.7
24	-11442.0	190.7
25	-11823.4	-381.4
26	-11251.3	572.1
27	-11632.7	0.0
28	-11251.3	190.7
29	-11251.3	190.7
30	-11442.0	0.0
31	-10869.9	0.0
32	-11060.6	381.4
33	-11251.3	190.7
34	-11251.3	190.7
35	-11632.7	-190.7
36	-11632.7	-190.7
37	-11251.3	190.7
38	-11442.0	0.0
39	5148.9	190.7
40	4767.5	190.7
41	5530.3	-190.7
42	4767.5	0.0
43	4767.5	0.0
44	4958.2	190.7
45	4767.5	0.0
46	5339.6	0.0
47	5148.9	190.7
48	5339.6	-190.7
49	4958.2	0.0
50	5339.6	-190.7
51	4767.5	190.7
52	5148.9	-190.7
53	5721.0	381.4
54	5148.9	-190.7
55	5339.6	0.0

time	deviation of calibrated cluster	deviation of not calibrated cluster
56	5339.6	0.0
57	4958.2	0.0
58	4958.2	190.7
59	5339.6	381.4
60	5721.0	190.7
61	5148.9	0.0
62	5530.3	190.7
63	5148.9	190.7
64	4576.8	0.0
65	5339.6	190.7
66	5148.9	-190.7
67	5148.9	190.7
68	5148.9	0.0
69	5530.3	-190.7
70	4958.2	190.7
71	0.0	0.0
72	0.0	-190.7
73	190.7	-190.7
74	0.0	-190.7
75	0.0	0.0
76	190.7	190.7
77	190.7	-572.1
78	381.4	190.7
79	0.0	572.1
80	-381.4	-190.7
81	-381.4	-190.7

Table B.5: External Synchronization: deviation after a change in the set of nodes used for internal clock synchronization at time 24 ($R_{ext.CS} = 32\ s$, $R_{measure} = 1\ s$, $g_{measure} = 190.7\ ns$). Values to Figure 7.7.

drift rate ($10^{-6}\frac{s}{s}$)	**frequency** absolute	relative (%)
36.0	0	0
36.5	94	0.14
37.0	96	0.15
37.5	348	0.53
38.0	447	0.68
38.5	436	0.67
39.0	856	1.31
39.5	1128	1.72
40.0	1573	2.40
40.5	3898	5.95
41.0	7234	11.04
41.5	8611	13.15
42.0	13889	21.20
42.5	10806	16.50
43.0	6052	9.24
43.5	3870	5.91
44.0	2072	3.16
44.5	1410	2.15
45.0	1233	1.88
45.5	496	0.76
46.0	392	0.60
46.5	219	0.33
47.0	100	0.15
47.5	137	0.21
48.0	40	0.06
48.5	0	0

Table B.1: Frequency of measured clock drift rate values of global time. Values to Figure 7.1.

deviation (TPU ticks, 1 TPU tick = 50 ns)	**frequency of deviation values** (%)
-18	0
-17	0.003
-16	0.004
-15	0.009
-14	0.008
-13	0.019
-12	0.039
-11	0.068
-10	0.151
-9	0.339
-8	0.612
-7	1.186
-6	2.153
-5	3.633
-4	5.511
-3	7.798
-2	10.061
-1	11.777
0	12.476
1	11.843
2	10.077
3	8.089
4	5.824
5	3.598
6	2.267
7	1.206
8	0.638
9	0.304
10	0.155
11	0.070
12	0.045
13	0.017
14	0.008
15	0.007
16	0.005
17	0.002
18	0

Table B.2: External synchronization: deviation of virtual clock time from relational clock time ($R_{ext.CS} = 16\ s$, $R_{measure} = 1\ s$, $g_{measure} = 1$ TPU tick length $= 50\ ns$). Values to Figure 7.4.

deviation (TPU ticks, 1 TPU tick = 50 ns)	**frequency of deviation values** (%)
-10	0
-9	0.001
-8	0.001
-7	0.005
-6	0.032
-5	0.277
-4	1.222
-3	4.299
-2	10.309
-1	18.072
0	23.207
1	20.665
2	13.387
3	6.012
4	1.940
5	0.445
6	0.097
7	0.025
8	0.004
9	0.001
10	0

Table B.3: Inter-cluster synchronization: deviation of virtual clock time from relational clock time ($R_{ext.CS} = 1\ s$, $R_{measure} = \frac{1}{16}\ s$, $g_{measure} = 50\ ns$). Values to Figure 7.5.

deviation (TPU ticks, 1 TPU tick = 190.7 ns)	**frequency of deviation values of**	
	calibrated cluster (%)	**not calibrated cluster** (%)
-8	0	0
-7	0.000	0
-6	0.016	0.005
-5	0.211	0.049
-4	1.404	0.421
-3	5.437	2.463
-2	13.524	9.926
-1	22.986	29.310
0	25.201	22.283
1	18.497	9.898
2	8.984	2.646
3	2.935	0.449
4	0.681	0.066
5	0.099	0.018
6	0.021	0.005
7	0.003	0
8	0	0

Table B.4: Frequency of deviation values. Values to Figure 7.6.

deviation (TPU ticks, 1 TPU tick = 50 ns)	**frequency of deviation values** (%) **history length**								
	1	2	4	8	16	32	64	128	256
-11	0	0	0	0	0	0	0	0	0
-10	0	0	0	0	0	0	0	0	0.001
-9	0.053	0	0	0	0	0	0	0	0.001
-8	0.099	0.030	0	0	0	0	0	0	0
-7	0.397	0.046	0	0	0.015	0	0.008	0	0
-6	1.007	0.183	0.023	0.046	0.084	0	0.030	0.023	0.023
-5	2.120	0.935	0.297	0.243	0.754	0.213	0.152	0.091	0.244
-4	4.217	2.738	1.529	1.369	2.720	1.255	1.110	0.730	1.623
-3	7.420	6.304	5.072	4.806	6.789	4.563	4.046	3.179	4.572
-2	11.599	12.251	11.878	10.973	12.716	11.240	10.395	8.760	10.272
-1	14.749	17.224	19.544	17.992	18.789	19.422	18.289	15.947	18.670
0	16.533	19.817	23.270	22.274	20.381	23.909	23.004	21.985	23.752
1	14.619	18.030	19.141	19.924	16.701	20.327	21.278	21.589	20.407
2	11.355	12.274	12.008	13.255	11.177	12.122	13.460	15.825	12.863
3	7.840	6.532	5.407	6.373	5.524	5.209	6.350	8.152	5.631
4	4.713	2.639	1.544	2.183	2.598	1.445	1.506	2.882	1.547
5	2.120	0.791	0.251	0.525	1.135	0.266	0.327	0.730	0.328
6	0.839	0.175	0.038	0.030	0.373	0.030	0.046	0.099	0.053
7	0.198	0.015	0	0	0.198	0	0	0.008	0
8	0.099	0.008	0	0	0.030	0	0	0	0
9	0.008	0.008	0	0	0.008	0	0	0	0
10	0.015	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0

Table B.6: Inter-cluster synchronization: deviation of virtual cluster time from relational clock time with changing history length ($R_{measure} = 1/16\ s$, $g_{measure} = 50\ ns$). Values to Figure 7.8.

deviation (TPU ticks, 1 tick = 50 ns)	**frequency of deviation values** (%) **history length**								
	1	2	4	8	16	32	64	128	256
-18	0.045	0	0	0	0	0	0	0	0
-17	0	0	0	0	0	0	0	0	0
-16	0.022	0.022	0	0	0	0	0	0	0
-15	0.112	0.022	0.022	0	0	0	0	0	0
-14	0.090	0.022	0.157	0	0	0	0	0	0.045
-13	0.270	0.157	0.067	0	0	0	0	0	0.045
-12	0.472	0.067	0.135	0	0.022	0	0.022	0	0.247
-11	0.944	0.315	0.202	0.135	0	0.045	0.067	0	0.315
-10	1.236	0.270	0.427	0.157	0.090	0.157	0.180	0.017	0.831
-9	1.798	0.584	0.449	0.404	0.270	0.292	0.517	0.028	1.101
-8	2.225	1.416	1.056	0.787	0.719	0.517	0.674	0.083	1.753
-7	3.213	2.112	1.978	1.551	1.236	1.079	1.303	0.155	3.056
-6	3.573	2.629	2.854	2.382	2.022	2.000	2.517	0.316	4.562
-5	4.787	4.427	4.202	5.056	3.393	3.820	3.618	0.870	6.247
-4	5.910	6.022	5.551	6.022	5.438	5.236	6.090	2.771	8.112
-3	6.427	8.112	8.315	7.933	7.888	8.292	8.157	5.742	10.989
-2	7.124	8.697	8.742	9.483	9.820	10.629	9.101	10.298	11.191
-1	7.079	9.169	11.258	10.562	12.022	12.202	11.124	16.462	11.371
0	8.315	10.135	10.517	11.640	12.629	12.921	12.337	12.100	11.146
1	7.775	10.360	10.337	10.494	11.910	11.056	12.022	18.801	8.315
2	6.966	9.618	8.202	9.416	10.494	10.315	10.225	14.134	7.371
3	6.135	7.865	7.753	7.573	8.135	7.775	8.360	8.785	4.831
4	5.955	5.416	6.045	6.270	6.067	5.865	6.315	4.767	2.921
5	5.348	5.213	4.157	4.315	3.708	3.551	3.843	2.228	1.910
6	4.067	2.899	2.809	3.079	2.225	2.090	1.730	1.352	1.303
7	3.663	1.618	1.596	1.371	1.213	0.966	0.944	0.715	0.742
8	2.247	1.281	1.169	0.674	0.404	0.764	0.494	0.216	0.517
9	1.573	0.719	0.831	0.494	0.202	0.315	0.225	0.105	0.292
10	0.854	0.292	0.404	0.180	0.067	0.090	0.090	0.028	0.225
11	0.966	0.292	0.225	0	0.022	0.022	0.022	0.022	0.135
12	0.360	0.180	0.157	0.022	0	0	0.022	0.006	0.157
13	0.202	0.022	0.180	0	0	0	0	0	0.090
14	0.112	0.022	0.090	0	0	0	0	0	0.090
15	0.090	0.022	0.045	0	0	0	0	0	0.067
16	0.022	0	0.067	0	0	0	0	0	0
17	0.022	0	0	0	0	0	0	0	0.022
18	0	0	0	0	0	0	0	0	0

Table B.7: External synchronization: deviation of virtual clock time from relational clock time with changing history length ($R_{measure} = 1\ s$, $g_{measure} = 50\ ns$). Values to Figure 7.9.

history length	**standard deviation** (TPU ticks; 1 TPU tick = 50 ns)
1	2.494
2	1.992
4	1.713
8	1.768
16	2.010
32	1.665
64	1.679
128	1.729
256	1.707

Table B.8: External synchronization: standard deviations of results presented in Figure 7.9. Values to Figure 7.10.

measurement interval (s)	**standard deviation of deviation values** (ns)					
	time difference measurement granularity (ns)					
	50	100	200	400	800	1600
2^{-4}	149.10	113.24	149.36	221.29	345.43	1107.03
2^{-3}	134.07	186.99	171.78	246.44	467.53	740.27
2^{-2}	170.04	172.77	222.54	251.84	349.91	1128.26
2^{-1}	235.90	246.11	205.96	309.53	409.15	597.55

Table B.9: Inter-cluster synchronization: range of deviation values ($H = 16$). Values to Figure 7.11.

measurement interval (s)	**range of deviation values** (ns)					
	time difference measurement granularity (ns)					
	50	100	200	400	800	1600
2^{-4}	900	1100	1200	1200	2400	3200
2^{-3}	950	1300	1200	1600	2400	4800
2^{-2}	1650	1400	1400	1600	3200	3200
2^{-1}	1700	2300	1600	2400	3200	6400

Table B.10: Inter-cluster synchronization: range of deviation values ($H = 16$). Values to Figure 7.12.

Curriculum Vitae

Michael Paulitsch

February 21, 1974	Born in Klagenfurt, Austria
September 1980 – July 1984	Elementary School in Pörtschach, Austria
September 1984 – July 1988	High School Bundesrealgymnasium Lerchenfeld, Klagenfurt, Austria
September 1988 – June 1993	State School of Engineering HTBLA Mössingerstraße 25, Klagenfurt, Austria
October 1993 – January 1999	Studies of Computer Science at the Technische Universität Wien, Vienna, Austria Graduation Computer Science with distinction
October 1993 – June 1999	Studies of Management Information Systems at the Technische Universität Wien, Vienna, Austria Graduation Management Information Systems
September 1996 – May 1997	Exchange and Undergraduate Student in Computer Science University of Illinois, Urbana-Champaign, IL, USA
September 1996 – August 1997	Research Assistant Center for Reliable and High-Performance Computing, University of Illinois, Urbana-Champaign, IL, USA
May 1997 – May 1998	Graduate Student in Computer Science University of Illinois, Urbana-Champaign, IL, USA
February 1999 – date	Research and Teaching Assistant Institut für Technische Informatik, Technische Universität Wien, Vienna, Austria
March 1999 – date	Doctoral Studies in Technical Sciences Technische Universität Wien, Vienna, Austria