MASTER'S THESIS

DESIGN, IMPLEMENTATION AND EVALUATION OF A HIGH AVAILABILITY SOLUTION FOR A LOGISTICS SYSTEM

submitted to

CAMPUS 02 GRAZ
FACHHOCHSCHULE DER WIRTSCHAFT

Degree program
Information Technologies & Business Informatics

By: Oleksandr Samoylyk
Personal identity number: 1410320015

Graz, 03. July 2017

....
Signature

DECLARATION OF HONOR

I declare on my honor that I have produced this thesis independently, that I have not used other than the mentioned sources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

....

Signature

ABSTRACT

Many areas of human activity demand high availability (HA) for the services provided by information technology (IT), including supply chain and especially logistics services as its backbone. Logistics automation has already been tightly linked with IT for some time now. Eventually, it is expected to reach a tipping point of digitalization under the Logistics 4.0 concept. Such anticipated convergence provided a basis for the thesis research. It was possible to apply time-proven approaches originally used in IT to produce a tailored and cost-effective HA solution for an IT-enabled logistics system in order to minimize costly downtime.

A multi-layer architectural pattern was adopted to focus the research on the opportunities to improve HA provided by innovative open-source software. “State-of-the-art” approaches, best practices, and challenges to attaining availability are covered in the course of the thesis. The case study was based on the efforts of a Styrian solution provider for intralogistics systems to improve and standardize HA solution for their software products and services. Mandatory HA requirements to ensure business continuity were used to outline a reference architecture for a generic HA solution by means of HA cluster for a logistics system. A prototype testbed, based on the relevant stack of technologies, validated and evaluated the proposed reference HA architecture.

The findings of this thesis demonstrate the feasibility of the idea to build a cost-optimized cluster-based HA solution using commercial off-the-shelf hardware and free open-source software which can deliver an improved level of availability for a modern logistics system.

KURZFASSUNG

In vielen Bereichen des täglichen Lebens erfordert es eine Hochverfügbarkeit (HA) von Services durch Informationstechnik (IT), dies inkludiert unteranderem auch Lieferketten in der Logistikbranche. Automatisierte Logistik ist bereits seit längerer Zeit eng mit IT verbunden und es wird erwartet, dass diese ihren „Tipping-Point“ der Digitalisierung durch das Organisationsgestaltungskonzept der Logistik 4.0 erreicht. Diese erwartete Konvergenz stellte die Grundlage dieser Masterarbeit dar. Es war möglich, bewährte Ansätze anzuwenden, welche ursprünglich in der IT genutzt werden, um eine zugeschnittene und kosteneffiziente HA-Lösungen für IT-fähigen Logistiksystemen zu schaffen um kostenintensive Ausfallzeiten zu minimieren.

Durch die Anwendung einer Schichtenarchitektur wurde der Forschungsfokus, auf Möglichkeiten zur Verbesserung der HA durch innovative Free/Libre Open Source Software gelegt. Im Rahmen dieser Masterarbeit werden „State-of-the-Art“-Ansätze, „Best Practices“ und andere Herausforderungen zur Erreichung von HA abgedeckt. Die Fallstudie basierte auf den Bestrebungen eines steirischen Herstellers für Intralogistiklösungen und Systeme im Bereich Lagerlogistik und Lagerautomation, zur Verbesserung und Standardisierung der HA-Lösung für ihre Softwareprodukte und -dienstleistungen. Um die betriebliches Kontinuität zu gewährleisten wurden notwendige Anforderungen an HA verwendet um eine technische Referenzarchitektur für einer generische HA-Lösung mit Hilfe eines HA-Clusters für ein Logistiksystem zu entwerfen. Durch Implementierung einer Prototype-Testumgebung, basierend auf dem relevanten Umfang von Technologien, wird die vorgeschlagene Referenzarchitektur geprüft und evaluiert.

Die Ergebnisse dieser Masterarbeit veranschaulichen die Durchführbarkeit einer kostenoptimierte Cluster-basierte HA-Architektur mittels handelsüblicher Hardware und Free/Libre Open Source Software welche eine Verbesserung der Hochverfügbarkeit von modernen Logistiksystemen ermöglicht.

CONTENTS

1 INTRODUCTION

"Anything that can go wrong will go wrong"
"The 4th Law of Thermodynamics" by Edward A. Murphy

Today, in the age of digitalization, we are witnessing a new industrial transformation – a shift from an electronic-based technologies to a smart automation, called Industry 4.0 (Ramsauer, 2013). Convergence between industry and IT synergizes this phase of industrial development (Nikolaus, 2013; ServTec Austria, 2015). A study conducted by Deloitte (2015) reveals the fact that warehousing and logistics are among those business segments which are seen at the very core of the digital transformation that occurs as part of Industry 4.0, while sales and services are the segments with the greatest potential to benefit from it. By studying the outcomes which follow Industry 4.0 along with Logistics 4.0 paradigms, Austrian Ministry for Transport, Innovation and Technology identified data availability as one of the promising areas of research (Bundesministeriums für Verkehr, Innovation und Technologie, 2015). The concept of data availability, as a valuable asset for business, and intralogistics, as "the backbone and enabler of Industry 4.0" (CeMAT, 2016b), narrowed down the research field and delimited the research topic of the thesis – the high availability (HA) of IT-based logistics systems.

1.1 Motivation

Companies are becoming increasingly dependent on their IT landscape, and its continuous availability is essential to organizational success. Hence, new challenges and requirements for IT as an enabler of business are constantly arising (CeMAT, 2016a; Hausladen, 2016). Modern-day solutions in the field of intralogistics are based on complex IT systems and have "little in common with the relatively one-dimensional storage and distribution of goods seen up until a few years ago" (Wolfenstein, 2015). Availability of logistics systems such as the high-bay warehouse management system has a direct impact on overall business continuity. They are supposed to run "24 hours a day, 7 days a week" in a seamless fashion for every stakeholder and cannot be stopped for a time period longer than a lunch break (Ronzon, 2016, pp. 11-12).

> *...many warehouses operate in 24/7 mode with three shifts per day. Availability is therefore crucial for supporting the business case for a warehouse management process control system. Any downtime disrupts supply chains, the state and operation of other systems, people, and so on, which ultimately means loss of business and money. Industrial automation systems in general, and process control systems specifically, therefore, typically demand a minimum availability of 99.999% — a maximum downtime of just over five minutes per year! (Buschmann, Henney, & Schmidt, 2007, p. 63)*

According to BCI's recent Supply Chain Resilience Report unplanned IT outages are the major source of supply chain disruption (64%), running ahead of extreme weather, earthquakes, product quality incidents, and transport network disruptions. Downtime of such mission-critical systems in B2B processes automatically implies the loss of productivity (58%) and revenue (38%), customer complaints (40%), and damage to company reputation (27%) (Business Continuity Institute, 2015). In some cases, extended downtime may even result in legal consequences (Sousa & Oz, 2015, p. 477).

Key findings of the recent surveys, yet unrelated to any company, demonstrate the vitality of high availability topics for business continuity:

- "*European businesses collectively suffer from almost 1 million hours of IT downtime each year (956,373 hours). That's an average of 14 hours per company per year.*" (CA Technologies, 2011)
- "*64% of enterprises surveyed experienced data loss or downtime in the last 12 months.*" (EMC, 2014)
- "*73% of the organizations have a service availability goal of over 99.91% (less than 8 hours of unplanned downtime a year) for mission critical systems.*" (Continuity Software, 2014)
- "*Estimated cost of downtime for small and medium-sized enterprises is between 20,000 EUR and 40,000 EUR per hour.*" (techconsult GmbH, 2013)
- "*57% of organizations have not calculated their hourly downtime costs after a failure.*" (Vision Solutions, 2015)
- "*13% of companies still do not have an HA solution.*" (Vision Solutions, 2016)
- "*Increased reliance on technology is being seen as a top risk which affects the availability of applications and services.*" (Forrester, 2013)
- "*Four nines – 99.99% uptime is now the minimum reliability required by 79% of organizations*" (ITIC, 2016)

Choosing a solution for high availability is no different from finding solutions in risk management – it is necessary to balance the marginal costs and the costs related to the risk of losses. The expenditures on availability should be justified by the expected cost of downtime. The primary goal is to find an optimal cutover point between the expected total cost of ownership (TCO) and benefits, which such solutions bring (KPMG, 2014). Figure 1-1 illustrates such a trade-off:

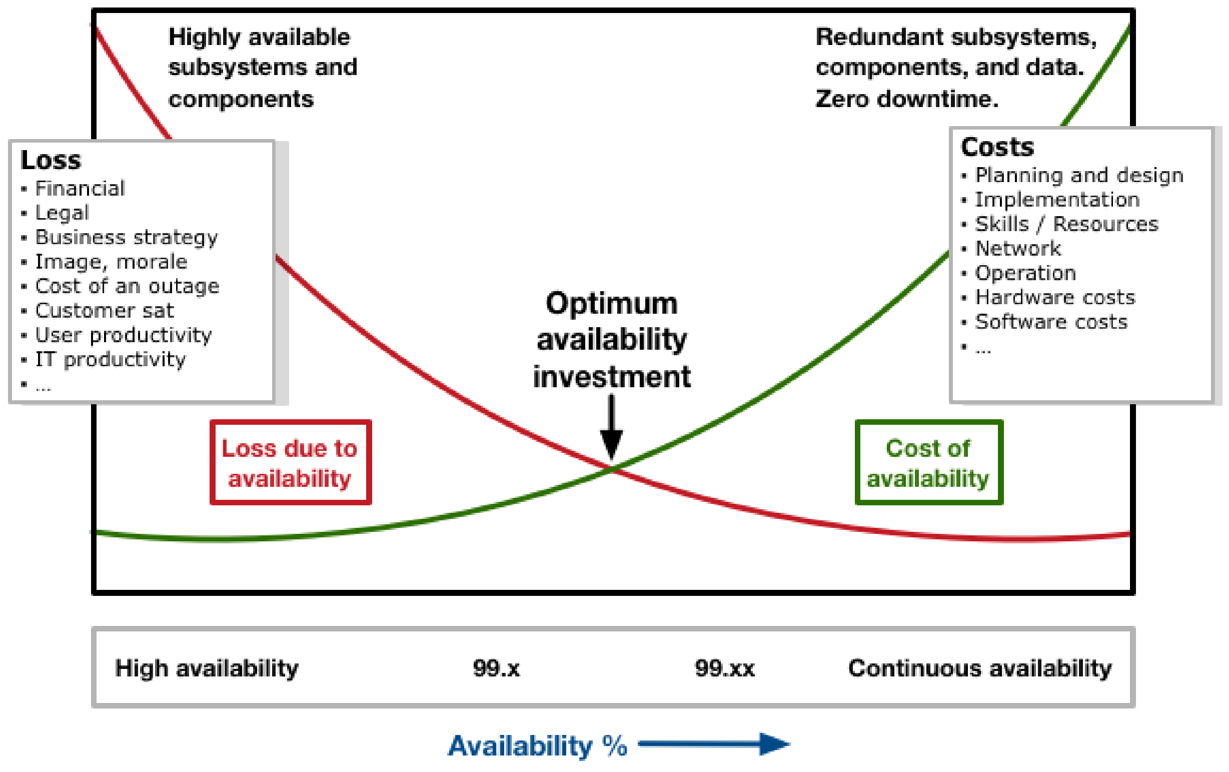


Figure 1-1: Costs versus benefit, based on (Zhu, et al., 2009, p. 12)

That is where a concrete solution for an abstract problem of high availability for a logistics system in cost-optimized way should be found. Currently, HA clustering has been considered one of the most optimal ways of solving problems for high available IT services (Critchley, 2015, p. 149; Forrester, 2014, p. 8). In this regard, the research conducted in the course of the thesis should the address a rising demand for designing, implementing, and evaluating cluster-based HA architecture for a logistics system that meets business objectives related to availability under real-world constraints of the intralogistics industry.

1.2 Aim and Objectives

Decomposition of the problem statement by adopting multi-layered architectural paradigm (infrastructure – platform – application) to HA clustering helped to refine and polish the operational research question as follows:

How can the availability of a logistics system be improved at application- and platform-layers, while reducing costs at the infrastructure-layer?

The research question is backed up by the idea of building a reliable system from unreliable components. (von Neumann, 1956)

The primary research question forms the basis for the following working hypotheses:

- *Approaches used to achieve and measure availability traditionally used for IT systems can be applied to a modern logistics system in a similar way.*

- *State-of-the-industry commercial off-the-shelf (COTS) hardware delivers an acceptable level of availability for a logistics system.*
- *Solutions for HA clustering based on free and open-source software (FOSS) are mature to be used under a mission-critical and real-time setting, such as warehouse automation.*

To test the working hypotheses and to give an answer to the research question an empirical research approach, with an overall aim of building a cluster-based HA solution for a logistics system is to be conducted.

First, the research process sets an objective to outline the reference architecture for HA clusters for logistics systems. This should be based on real-world scenarios, but still be abstract enough to be applicable to various similar business scenarios.

Furthermore, prototype implementation of the reference architecture should utilize current best-of-breed technology mix and, when applicable, FOSS components.

Finally, findings of the thesis are to be of a practical significance for KNAPP[1] Company – all-in-one solution provider of customized intralogistics systems. The company's needs in optimizing and standardizing HA solutions are aligned with technical assessment, which builds a basis for applied research.

1.3 Scope

The main motivation of the master's thesis is attaining high availability for a logistics system and how certain technical implementation of that system can be achieved. Therefore, primary focus is on those baselines that are necessary to understand and to answer the research question. Accordingly, such topics as:

- business continuity
- information security
- risk management
- total cost of ownership
- return on investment

which should also be involved in the process of implementing HA for a logistics system, but which are not explicitly assessed in the course of this thesis.

It should also be noted, that a logistics system is being viewed as an existing real-word software application, designed to support warehouse and distribution operations. Unlike general purpose software, this category of applications is extremely customized and fully integrated proprietary software that is sold as highly tailored turnkey solutions to specific business needs (Klappich,

[1] https://www.knapp.com/ – KNAPP - warehouse logistics solutions

2013; Software Advice, 2015). This fact limits research design to a single-case study of KNAPP KiSoft[2] software, the only software that was provided for the purposes of empirical research.

1.4 Research Design

This thesis follows an empirical research approach. Firstly, it contains an interdisciplinary literature review, comprised of the body of knowledge from such domains as systems engineering, computer science, software engineering and industrial engineering. This literature review is conducted to tackle the research question. This process provides the theoretical and methodological basis for the experimental study. Review of “state-of-the-art” HA techniques and principles offers insight into a good system design. Next, within the real-world situation based on failure scenarios along with HA objectives and requirements for the investigated logistics system are examined. The output from the empirical inquiry is used to outline a possible generic approach in the form of reference HA architecture that deals with the problem statement. In order to conduct an experiment a prototype implementation of reference HA architecture is made. A testbed platform prototype is set up to test constructed hypotheses. Finally, the results that derived from the applied research methods is analyzed and presented.

1.5 Related Work

The literature review process revealed two streams of work related to the specified problem domain: academic papers and industry-backed publications. Recent academic papers on Computer Science and Engineering are mostly focused on high availability for web-oriented applications (Moniruzzaman & Hossain, 2014), database systems (Kim, Salem, Daudjee, Aboulnaga, & Pan, 2015) or contextless high availability in cloud computing (Kanso & Lemieux, 2013; Colman-Meixner, Develder, Tornatore, & Mukherjee, 2016) and on concept of virtualization (Calzolari, et al., 2010; Li, Kanso, & Gherbi, 2015), as its enabler. By contrast, industry-related studies including that of Schulze (2007) and Maier (2011) miss an IT aspect for high availability of logistics systems. Finally, studies conducted by Gunasekaran (2007) and Hausladen (2010) examine a framework of an IT-based logistics system and leave out observations of non-functional characteristics such as availability.

In contrast, Furmans, Nobbe, & Schwab (2011) named high availability along with flexibility and configurability, as inherent features of effective modern logistics systems. Correspondingly, Buschmann, Henney, & Schmidt (2007) stated that attainment of high availability for logistics systems, particularly warehouse automation systems is one of the most complex and challenging tasks that can be addressed by distributed computing.

[2] https://www.knapp.com/en/solutions/technologies/software/ – KNAPP group’s software product line

A series of related industry analysis reports, published by Supply Chain Digest (2013) and O'Brien (2017) suggest that current cloud-based offerings such SaaS (software as a service), PaaS (platform as a service), and IaaS (infrastructure as a service) can often adequately address high availability requirements of IT-based logistics systems. Nevertheless, the authors depict constraints for moving into cloud for a class of logistics systems based on soft real-time requirements. For example, one system that demands predictable response times is WCS. WCS serves as logistics "middleware" between external software (e.g. ERP, WMS) and various equipment controllers (e.g. PLC) that coordinate automated warehouse activities (Son, Chan, Choi, Kim, & Higuera, 2015). Therefore, unlike WMS or ERP systems, WCS is always installed locally at the warehouse in order to eliminate higher network latency over the WAN which negatively impacts the warehouse performance and to be able to maintain warehouse operations in case of network outage between the warehouse and the corporate WMS/ERP system, which is usually located off-site. Other burdens impacting HA for hosted logistics system in cloud environments include business concerns about security in public clouds, as such systems contain sensitive information, on a par with "overhead" expenses for ad hoc deployment of private cloud as part of intralogistics turnkey solutions. (Supply Chain Digest, 2013; O'Brien, 2017)

Finally, Ronzon (2016) considered retrofitting of high availability into an existing legacy logistics systems that are no longer maintainable as another approach to achieve the desired level of availability. Virtualization-based solutions are seen as a feasible strategy to overcome the difficulties, risks, and financial expense introduced by using such an approach.

1.6 Thesis Outline

Master's thesis consists of seven chapters subdivided into sections and subsections.

Initial sections of the first chapter introduce the topic of the thesis, its relevance, practical value and the necessity for solution. The research question, delimitations and research methods are formulated in following sections. Further section highlights related works in the context of high availability topic for logistics systems. The summarized structure of the thesis completes the chapter.

The second chapter presents logistics system taxonomy and explains essential relevant theory that underpins core availability concepts, basis principles, influencing factors and measurement methods.

The third chapter captures a state of research by presenting underlying approaches, technologies, industry best practices, and challenges related to the topic of the thesis. The final sections of the chapter are dedicated to architectural patterns that can be applied to provide increased resilience for logistics systems.

The fourth chapter depicts approach which is applied to get integrated HA solution for a logistics system. It opens with a situation analysis to gather real-world requirements and define objectives for a potential solution. Finally, based on objectives and requirements from use case scenario a proposed reference architecture in the form of shared-nothing failover cluster is outlined.

The fifth chapter is dedicated to an experiential setup of HA cluster. The first sections of the chapter present the process of mapping the defined reference architecture to the porotype implementation of HA cluster. It is followed by description of testbed and list of conducted experiments. Next, provides results derived from experimental evaluation are provided. At the end of chapter, evaluation of prototype testbed is done.

The sixth chapter presents the findings of the conducted study.

The final chapter summarizes the most significant points of the research and answer the research question based upon the outcomes of the study and provides recommendations for the further research.

2 BACKGROUND AND RELEVANT THEORY

To establish underlying understanding of the thesis topic this chapter provides a brief introduction to the taxonomy of IT-based logistics systems, prerequisite concepts of availability and reliability along with their measurement methods.

2.1 Logistics Systems

IT-based logistics systems are being subject to broad research in computer science, information systems, and service science (Leukel, Ludwig, & Norta, 2011, p. 211). According to (Wang & Pettit (2016) IT-based logistics system is defined as an umbrella term that has a range of software implementations depending on a specific field of logistics. It is proposed to adopt a best-of-breed system for a certain logistics activity, as it is very complicated to build a "one-size-fits-all" solution to cover all logistics activities within a company. (Wang & Pettit, 2016, p. 6)

Authors Hausladen & Haas (2016) and Reji (2008) described logistics systems as a combination of hardware and IT-enabled solutions to manage, control, and measure the logistics activities. Hardware solutions include computers, I/O devices, and storage media, whereas IT-enabled solutions encompass all kinds of software applications (e.g. warehouse logistics systems), technological approaches (e.g. use of RFID, EDI, business intelligence), and concepts (e.g. big data implementation strategies or the introduction of cyber-physical systems in production or distribution) which support logistics and supply chain processes. (Hausladen & Haas, 2016, p. 131; Reji, 2008, p. 321)

The work of Hausladen (2010) outlined a reference model for an IT-based logistics system using the production system approach. According to this model an IT-based logistics system can either be a leadership or a performance system. Performance systems encompass logistics-specific applications, which are connected to a leadership system. In its turn, the leadership system comprises relevant subsystems such as the planning, organization, information, control, human resources and coordinate performance system.

Table 2-1 outlines the main fields of logistics and corresponding IT-based logistics applications:

Logistics Activity	Applications
Supply Chain	▪ Production Planning and Control Systems (PPC) ▪ Enterprise Resource Planning Systems (ERP) ▪ Advanced Planning Scheduling Systems (APS) ▪ Supply Chain Management Systems (SCM Systems) ▪ Order Processing Tools ▪ Transportation Management Systems (TMS)

Warehouse Logistics	▪ Warehouse Management Systems (WMS) ▪ Warehouse Control Systems (WCS) ▪ Vendor Managed Inventory (VMI) ▪ Merchandise Information Systems (MIS) ▪ Cross Docking ▪ eConsignment ▪ Robogistics
Procurement Logistics	▪ Supplier/Demand Catalogues ▪ Supplier E-Kanban ▪ Online Auctions ▪ Virtual Marketplaces
Production Logistics	▪ Just-in-Time (JIT) ▪ Just-in-Sequence (JIS) ▪ Production E-Kanban ▪ Digital Factory/Virtual Logistics
Maintenance Logistics	▪ Computerized Maintenance Management Systems (CMMS) ▪ Supervisory Control and Data Acquisition Systems (SCADA) ▪ Condition Monitoring Systems ▪ Maintenance Platforms
(Re-)Distribution Logistics Subsystems	▪ Efficient Consumer Response (ECR) ▪ Customer Relationship Management (CRM) ▪ Logistics Platforms ▪ Tour Planning & Route Optimization ▪ Tracking & Tracing ▪ Telematics ▪ Milk Run ▪ ePayment ▪ Last Mile Logistics

Table 2-1: IT-based logistics systems and their corresponding applications (Hausladen, 2010, p. 242)

2.2 Availability Concepts and Principles

This section describes the interrelated concepts of availability and reliability, along with opposite concept of unavailability known as outage.

2.2.1 Availability and Reliability

There are two core concepts which underlie this thesis: *availability* and *reliability*.

IEEE Standard Glossary of Software Engineering Terminology defined the terms "availability" and "reliability" as follows:

Availability: "*the degree to which a system or component is operational and accessible when required for use.*" (IEEE, 1990, p. 11)

Reliability: "*the ability of a system or component to perform its required functions under stated conditions for a specified period of time*". (IEEE, 1990, p. 62)

The work of Algirdas, Laprie, Randell, & Landwehr (2004) described availability as *"readiness for correct service"*, whereas reliability as *"continuity of correct service."* These are significant, because they are foremost among the attributes that comprise the concept of "*dependability*". (Algirdas, Laprie, Randell, & Landwehr, 2004, p. 3)

Further, reliability represents a *property* of system component, whereas availability characterizes its *state*. For example, a system can go down just for a millisecond every hour, thus being highly available, but still highly unreliable. On the other hand, a system that never fails, but is shut down for a long period, is highly reliable, but not highly available. (Tanenbaum & van Steen, 2003, pp. 322-323)

It is also important to understand the concepts of uptime and downtime, as it relates to reliability and availability. The total time spent in an operative state is called uptime, while inoperative state is called downtime (Pall, 1987, p. 65). Therefore, statistically system availability can be quantified by using a generalized ratio function of the uptime to the sum of uptime and downtime and can be expressed using the following equation (Gransberg, Popescu, & Ryan, 2006, p. 247):

$$Availability = \frac{Uptime}{(Uptime + Dowtime)}$$

Equation 2-1: Generalized time-based availability formula (Gransberg, Popescu, & Ryan, 2006, p. 247)

The values of uptime and downtime can be either predicted by using the mathematical modeling techniques (e.g. Markov availability model,) or derived from actual field measurements (e.g. from outage trouble tickets or via service probes) (Bauer & Adams, 2012, p. 35).

2.2.2 Outages

Outages are the occurrences that impact availability. Similar to the concept of downtime, an outage is defined as the period of time when a total loss of function occurs. As opposed to downtime, outage can encompass a partial loss of function, rather than a total loss of function (Stapelberg, 2009, p. 405).

There are two types of outages: *planned* and *unplanned.* A planned outage is prepared and scheduled during the maintenance window, and is typically caused by software updates or hardware upgrades, whereas unplanned outage is unpredictable, typically caused by hardware, software, or network failure, environmental problem, and other failures. (Kyne, et al., 2014)

By analyzing about 2300 outages over a span of more than 11 years in the Cancer Research Center (FHCRC)[3] Kendrick (2012) came to the conclusion that more than a half of outages were a result of planned maintenance. The ratio between planned outages and unplanned outages was 55% to 45%.

As for common reasons for unplanned outages, a recent survey, conducted among 3.300 IT decision makers from mid-size to enterprise-class businesses across 24 countries, showed that 53% of outages were caused by hardware failure, followed by outages caused by loss of power (39%), and software failure (38%) (EMC, 2014, p. 23). Figure 2-1 presents the results of this survey:

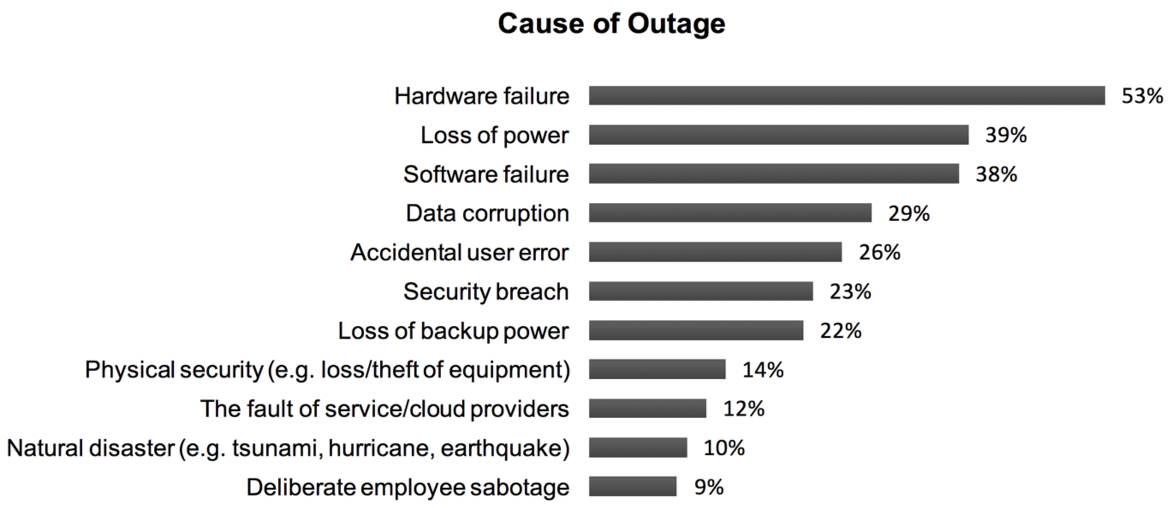


Figure 2-1: Causes of outages, based on (EMC, 2014, p. 23)

2.3 Measuring High Availability

This section introduces the main characteristics that define the requirements for the availability concept of data and IT services.

[3] https://www.fredhutch.org/en.html – Fred Hutchinson Cancer Research Center

2.3.1 Recovery Metrics

Development of technical solutions for the protection of business-critical data and services usually involves two basic concepts derived from a business impact analysis (BIA): *recovery time objectives* and *recovery point objectives*.

ITIL v3 defines these two objectives as follows (AXELOS, 2011):

Recovery time objectives (RTO) are "*the maximum time allowed for the recovery of an IT service following an interruption*".

Recovery point objectives (RPO) are "*the maximum amount of data that may be lost when service is restored after an interruption*".

Both, RTO and RPO, are measured in terms of time, or at *what speed* (RTO) and to *what point* (RPO) (Critchley, 2015, p. 311). Ideally, for most critical processes they should be brought to zero. However, in real life, these two objectives are balanced against each other to optimize the cost/benefit ratio (Rogers, et al., 2011, p. 297). Since the relationship between financial investments and RTO/RPO is non-linear, but rather exponential, RTO and RPO are defined for each business-critical system individually, taking into account its business impact (Allspaw & Robbins, 2010, p. 229). Figure 2-2 depicts the typically achievable recovery objectives along with the means to achieve them:

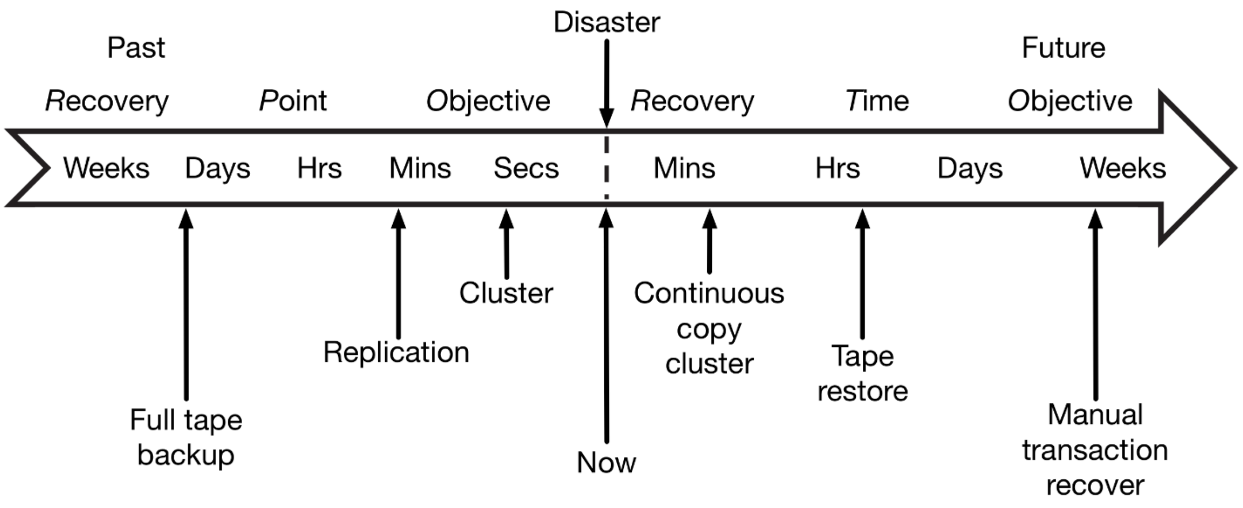


Figure 2-2: Recovery objectives: RTO and RPO, based on (Critchley, 2015, p. 322)

2.3.2 Reliability Metrics

There are various measures characterizing the dependability of computer systems. First, it should be understood that four major reliability parameters influence availability: *mean time between failures*, *mean time to failure*, *mean time to diagnose*, and *mean time to repair* (Koren & Mani Krishna, 2010, pp. 5-6). They can be defined as follows (Castano & Schagaev, 2015, pp. 18-20):

Mean time between failures (MTBF): the average time that the system runs between failures.

Mean time to failure (MTTF): the average time until a failure occurs.

Mean time to diagnose (MTTD): the average time required to diagnose a failure.

Mean time to repair (MTTR): the average time required to fix a system.

Hence, MTBF can be expressed as combination of three other parameters ($\mathrm{MTBF} = \mathrm{MTTF} + \mathrm{MTTD} + \mathrm{MTTR}$). Their relationship with one another are depicted in Figure 2-3:

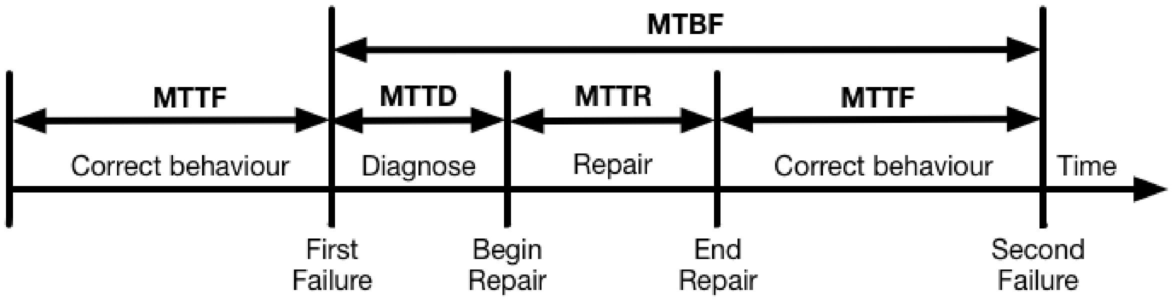


Figure 2-3: Parameters influencing availability, based on (Allspaw & Robbins, 2010, p. 83)

2.3.3 Availability Metrics

Now, by using reliability parameters the general formula for time-based availability (Equation 2-1) may be written as:

$$A = \frac{MTTF}{MTTF + (MTTD + MTTR)} = \frac{MTTF}{MTBF}$$

Equation 2-2: Time-based availability formula, as mentioned by Allspaw & Robbins (2010, p. 83)

Hence, in order to increase availability ($A \to 1$) it is necessary to increase MTTF ($MTTF \to \infty$) and decrease MTTD ($MTTD \to 0$) and MTTR ($MTTR \to 0$). In the context of achieving high availability, this intention raises a practice-relevant dichotomy, related to whether shorter outages or fewer total outages are preferable. (Allspaw & Robbins, 2010, pp. 83-85)

According to Schwartz (2015) a short outage (lower MTTR) is not always preferable over rare, longer outages (higher MTBF). Some systems can be affected by a cascade effect when they constantly go down and recover quickly which leads to a prolonged “warm-up” time (e.g. re-establishing TCP connections, destroying and recreating processes or state).

Relatedly, Franke (2012) proposed the following strategy to choose between many short or fewer long outages: “when outage costs are proportional to outage duration, more but shorter outages should be preferred to fewer but longer, in order to minimize variance” (Franke, 2012, p. 22).

Time-based availability metrics are a commonly used approach to measure availability of systems and services. However, in some cases, defining availability in terms of the request success rate (e.g. proportion of successful requests over a specific time window) might be more appropriate. (Beyer, Jones, Petoff, & Murphy, 2016, p. 27)

Availability metrics are the most widely used framework for key indicators (KIs) to be met as part of service-level agreement (SLA) (Hajinazari & Abbas, 2012). Consequently, availability classification may provide a discrete structure to choose an appropriate high availability solution that fulfills that KIs. This indicates that there are several approaches to classify availability and to define high availability as part of that classification, including both numerical-based (Gray & Siewiorek, 1991) and multilevel-based options (HRG, 2003; IDC, 2013a).

The work of Gray & Siewiorek (1991) introduced the following equation to derive the availability class of computer systems:

$$Availability\ Class = \log_{10}\left(\frac{1}{1 - Availability}\right)$$

Equation 2-3: Derivation of availability class (Gray & Siewiorek, 1991, p. 40)

Table 2-2 tabulates availability of typical system classes:

System Type	**Unavailability (minutes/years)**	**Availability (%)**	**Availability Class**
Unmanaged	50000	90	1
Managed	5000	99	2
Well-managed	500	99.9	3
Fault-tolerant	50	99.99	4
High-availability	5	99.999	5
Very-high-availability	0.5	99.9999	6
Ultra-availability	0.05	99.99999	7

Table 2-2: Classes of system availability (Gray & Siewiorek, 1991, p. 40)

Further, the Harvard Research Group[4] identified five classes within their Availability Environment Classification (AEC). Their classification is defined in terms of the impact of an outage on both, the activity of the business and the end user of the service, along with the data availability. The class levels are cumulative, so each successive level includes all the functionality of the previous level. Table 2-3 explains each Availability Environment in detail (HRG, 2003):

HRG Class	**Name**	**Explanation**
AEC-0	Conventional	Business functions that can be interrupted and where the availability of the data is not essential. To the user work stops and uncontrolled shutdown occurs. Data may be lost or corrupted.
AEC-1	Highly Reliable	Business functions that can be interrupted as long as the availability of the data is insured. To the user work stops and an uncontrolled shutdown occurs. However, data availability is ensured. A backup copy of

[4] http://www.hrgresearch.com/ – Harvard Research Group, Inc. (HRG)

		data is available on a redundant disk and a log-based or journal file system is being used for identification and recovery of incomplete transactions.
AEC-2	High Availability	Business functions that allow minimally interrupted computing services, either during essential time periods, or during most hours of the day and most days of the week throughout the year. This means the user will be interrupted but can quickly relog on. However, they may have to rerun some transactions from journal files and they may experience some performance degradation.
AEC-3	Fault Resilient	Business functions that require uninterrupted computing services, either during essential time periods, or during most hours of the day and most days of the week throughout the year. This means that the user stays online. However, the current transaction may need restarting and users may experience some performance degradation.
AEC-4	Fault Tolerant	Business functions that demand continuous computing and where any failure is transparent to the user. This means no interruption of work; no transactions lost; no degradation in performance; and continuous 24x7 operation.

Table 2-3: HRG Availability Environment Classifications (HRG, 2003)

Another, more sophisticated, approach was used by the analyst firm IDC[5] to describe multiple levels of availability. This combined approach is based on resilience capabilities as well as impact of component failure on end user. Table 2-4 differentiates four availability levels within IDC's availability framework (IDC, 2013a):

Availability Level	**Characterization**	**Impact of Component Failure**	**System Protection Factor**
Availability level 1 (AL1)	Not shipped as highly available	Need to switch to redundant resources before processing resumes	No special protection for availability
Availability level 2 (AL2)	Workload balancing	Balancing may not be perceptible to end users because of retry	User request is redirected to alternate resources

[5] https://www.idc.com/ – International Data Corporation

Availability level 3 (AL3)	Clustered server	Short outage is needed for failover to take place	User workload fails over to alternate resources
Availability level 4 (AL4)	Fault-tolerant server	Switch to alternate resources is not perceptible to end users	100% component and functional resiliency

Table 2-4: IDC's availability spectrum (IDC, 2013a, p. 13)

3 GENERAL SOLUTION SPACE FOR HIGH AVAILABILITY

"Any sufficiently advanced technology is indistinguishable from magic."
"Third Law" by Arthur C. Clarke
"Any technology distinguishable from magic is insufficiently advanced."
by Barry Gehm

Based on the literature review, certain gaps in research begin to appear. As such, the current chapter presents "state-of-the-art" approaches and architecture design patterns which can be used to attain high availability.

3.1 Means to Attain High Availability

Possible ways to achieve high availability are discussed in detail in this section. They serve as a point of reference for future empirical research, and work to create a framework for exploring the gaps within the current body of research.

3.1.1 Hardware Availability

Independent ITIC 2016-2017 Hardware Reliability survey showed that *"45% of respondents rely on the built-in redundant hardware capabilities of their servers to provide high availability and failover protection*" (ITIC, 2016). Additionally, two surveys, conducted by EMC[6] and Continuity Software[7], listed hardware failure first among of the most common reasons for outages (Continuity Software, 2014, p. 16; EMC, 2014, p. 23). Therefore, improving hardware availability is among the most prevalent breach issues that need to be addressed.

Moreover, true continuous availability is attainable only in situations where at any given time an exact copy of the server with a running service exists. Creating a copy after a hardware failure takes time and, therefore, causes an interruption of supplying that a service. In addition, after a failure the contents of volatile RAM, or of a failed server, is unavailable, which leads to data loss. To overcome this problem a *redundant hardware computing* approach is widely-adopted for mission-critical systems in various fields from banking to the healthcare industry (Gainaru & Cappello, 2015, pp. 104-106). The basic architectural design for such an approach implies a concept of splitting the resources of a server where major components in the system (e.g. CPUs, memory, peripheral controllers) are duplicated, and the computations are performed simultaneously and independently on a separate companion unit. Comparator checks the output of these units and in case of discrepancy, an error detection and a corresponding attempt to correct that error can be

[6] http://www.emc.com/ – EMC Corporation

[7] http://www.continuitysoftware.com/ – Continuity Software: IT Operations Analytics

performed. If an error is uncorrectable, then the defective component is switched off. (Lee & Anderson, 1990, pp. 92-93)

Within computer hardware engineering, the acronym *RAS* (Reliability, Availability, and Serviceability) is used to describe a set of mainly hardware-related robust features that enhance data protection and provide a higher level of availability. Initially, it was IBM mainframes[8] that possessed RAS features (Siewiorek & Swarz, 2014). High-end enterprise mainframes are still available on the market and dominate the server landscape of Global Fortune 500 companies by 71% (SHARE, 2013). Later, appearing in the late 1970s, servers were based on a reduced instruction set computing (RISC) processor design based on the early 1990s mainframe-inspired RAS functionalities. Until the mid-2000s proprietary RISC-based (e.g. IBM Power and Solaris SPARC systems) servers were traditionally chosen for "always-on" "always-available" enterprise services as they were superior to other architecture in terms of performance and availability (Bach, 2014, pp. 18-20; Intel, 2005, p. 5). However, the situation has changed, as modern commodity off-the-shelf servers based on open, industry-standard x86 architecture, offer a cost-effective alternative to RISC-based servers and can be readily used for mission-critical computing and attain 99.999% uptime (ITIC, 2017, p. 14). Table 3-1 lists most common RAS capabilities available on a modern COTS x86-based server hardware:

RAS Capability	Explanation
Lockstep	Identical components of the system run in parallel the same set of instructions. Each of the components is an active spare. Thus, if one of them fails, the other continues operation as usual, without any interruption or data loss.
Hot swapping and hot plugging	Ability to replace, add or remove components, such as hard disk drives, cooling fans, CPUs and memory without powering down the server
Machine Check Architecture (MCA)	Hardware errors (e.g. bus errors, ECC errors, parity errors, cache errors) are reported to the operating system thus allowing the operating system to perform corrective action and continue to work even after error detection.
Error-correcting codes (ECC)	Traditional ECC technologies perform single-bit error checking to detect and correct data corruption. Advanced ECC technologies can correct multiple bit errors as well.

[8] http://www.ibm.com/ibm/history/exhibits/mainframe/mainframe_intro.html – IBM Mainframes

Memory mirroring	Each CPU simultaneously uses two memory modules for data write and read. If one memory module fails, CPU can still access memory, since in other available memory module contains the valid data.
Intelligent Platform Management Interface (IPMI)	IPMI provides autonomous monitoring and management capabilities built directly into server hardware and firmware.
Power-on self-test (POST) and built-in self-test (BIST)	The ability of the server system and its individual components (e.g. controllers, power supplies, sensors, etc.) to perform checks their operability (during system initialization or periodically).
Watchdog timer	Hardware timer to detect a system hang and, if necessary, force its restart.

Table 3-1: RAS features of modern x86-based servers, based on (DELL, 2016; HP, 2013, p. 8; Intel, 2011, pp. 12-15; Lenovo, 2016)

According to Critchley (2015), "*it is now possible with current vendor hardware, software (operating systems and hypervisors), and services, along with third-party software, to achieve the manageability and availability of the managed mainframe of the good old days*" (Critchley, 2015, p. 40).

3.1.2 Storage Availability

Data availability is dependent on the reliability and availability of its underlying data storage. Magnetic storage media, primarily hard disk drives, are the most widely used technology for data storage (IDC, 2013c). Maintenance of storage reliability is an important applied academic problem and its solving complexity increases with the amount of drives and their individual capacities (Elerath & Shah, 2004; Ramabhadran & Pasquale, 2006; Xin, Schwarz, & Miller, 2005).

Studies on issues of hard drive reliability are being conducted for decades. A Google-enabled study based on more than 100.000 hard drive disk samples shows annual failure rates (AFR) for individual hard drives varies from 1.7% for drives in their first year of operation to 8.6% for three-year-old drives (Pinheiro, Weber, & Barroso, 2007, p. 4). Similar findings were gained by an academic study carried out at Carnegie Mellon University covering in total a population of more than 100.000 hard drives from at least four different vendors (Schroeder & Gibson, 2007). Disk failure data were collected from several large high-performance computing systems and internet service sites over a 5-year timespan. It turned out that AFRs typically exceeded 1%, with 2-4% commonly occurring and up to 13% overall. According to Schroeder & Gibson (2007) the failure rate for hard drives is not constant during the operating time. There was an early start of hard drive degradation ("infant mortality") and disk replacement rate grew steadily with the increase of years, although

Yang & Sun (1999) assumed that this effect should not occur until a nominal lifetime of 5 years. Moreover, the field replacement practices for hard drives (Schroeder & Gibson, 2007) revealed that MTBF rates in the datasheets from hard drive manufacturers are set too high. In its turn, recent statistics data provided by Backblaze (2017) with a sample population of 82.516 hard drives over a span of four years reveal an AFR of 2.07%. In this case, in contrary to Schroeder & Gibson (2007), a so-called "bathtub curve," or a lifecycle failure pattern typical for other hardware components, could be yielded for hard drives. It indicates a failure rate of around 5% within the first 18 months (early failures), then AFR constantly drops (random failures) to less than 1% and starts to grow up to 17% after about three years in use (wear-out failures).

Use of multiple hard drives was an early attempt to improve reliability of data storage in case of a disk failure. Namely, Patterson, Gibson, & Katz (1988) described such an approach for combining a few cheap hard drives into a single logical device to improve the capacity and I/O performance of a storage system, where the failure of some drives does not result in the failure of the entire storage system. This data storage virtualization technology is known as *RAID* (redundant array of independent disks). RAID technology became widely used, resulting in most modern server systems being equipped with RAID controllers (Pearl, 2015, p. 288). Table 3-2 describes standard RAID levels in terms of availability and redundancy capabilities:

RAID level	**Availability feature**	**Redundancy and other features**
RAID 0	Low level of availability as a disk failure causes total loss of its data	▪ No redundancy ▪ Cheapest RAID configuration
RAID 1	High level of availability as a disk failure can be replaced by its mirrored disk	▪ Data in one disk belonging to this configuration is completely mirrored ▪ Fast reads and slower writes ▪ Very expensive RAID configuration
RAID 2	High level of availability as a disk can be replaced by multiple disks consisting of its data	▪ Bit-level data are striped (distributed data segments) across various disks ▪ Parity information is stored on a dedicated parity drive ▪ Multiple disks are required for read and write
RAID 3	High level of availability as a disk can be replaced by multiple disks consisting of its data	▪ Byte-level data are striped (distributed data segments) across various disks ▪ Parity information is stored on a dedicated parity drive

		▪ Multiple disks are required for read and write
RAID 4	High level of availability	▪ Block-level data are striped (distributed data segments) across various disks ▪ Parity information is stored on a dedicated parity drive which becomes bottleneck for writes
RAID 5	High level of availability	▪ Block-level data are striped (distributed data segments) across various disks ▪ Parity information is stored on a multiple parity drive which eliminates bottleneck for writes
RAID 6	Very high availability	▪ Block-level data is striped (distributed data segments) across various disks ▪ It employs $P + Q$ redundancy to protect against two disk failures

Table 3-2: Standard RAID levels and availability features, (Shivakumar, 2014, p. 76)

Furthermore, modern RAID controllers enhance the means to improve resilience of storage system by taking advantage of using nested combinations of standard RAID levels (e.g. RAID 1+0, RAID 0+1), as well as utilizing hot spare disks to replace failed disks in array “on-the-fly”, along with a battery backup unit (BBU) to protect cached data (e.g. pending writes) in case of power outage. (Schwartz, Zaitsev, & Tkachenko, 2012, pp. 414-418)

Besides hardware RAID, a number software-based implementations provide RAID functionality. Such implementations can be done within operating system (e.g. as a virtual logical device by md driver in Linux kernel[9]), within logical volume manager (e.g. LVM2[10]), or as a part of “next-generation” file system (e.g. ZFS[11], Btrfs[12]) (Critchley, 2015, p. 85).

Nevertheless, cloud computing and the big data paradigm shift has been changing the traditional approaches for storage systems (Juve, et al., 2009; Das, Agrawal, & Abbadi, 2010; Ko, Hoque, Cho, & Gupta, 2010). Traditional RAID strategies are unable to cope with such exploding data

[9] https://raid.wiki.kernel.org/ – Linux Raid

[10] http://www.sourceware.org/lvm2/ – Logical Volume Manager

[11] http://www.open-zfs.org/ – The OpenZFS Project

[12] https://btrfs.wiki.kernel.org/ – Btrfs Official Website

volumes (Jewell, et al., 2014, pp. 8-9). One of these limitations in the area of large data volumes is imposed by RAID rebuild time. Whereas the recovery time depends on drive capacity, the bigger the drive, the longer it takes to rebuild the array. According to (Shenoy, 2015, p. 10) RAID rebuild times for high-capacity drives (8TB+) may take up days or even weeks. Subsequently, this increases the probability of another failure during rebuild process which will result in data loss. Thus, with the growing capacities of the individual disks and storages, RAID reliability decreases (Siewert & Scott, 2011; Intel, 2012).

In order to overcome the shortcomings of using RAID for very large data sets "post-RAID" or "noRAID" approaches were introduced (Harris, 2012). They are based on *erasure coding* to break the data segments into fragments that are encoded, mainly using the Reed-Solomon coding algorithm, and stored across different devices with an arbitrary number of redundant pieces of data (Plank, 2013). Such an approach is being adopted by clouding computing (Khan, Burns, Plank, & Pierce, 2012), distributed storage (Antony, et al., 2016, pp. 39-41; Ford, et al., 2010), and software-defined storage (Singh, 2016, pp. 224-232).

Lastly, there are data corruptions that are not detected at the RAID level. Work of Chen, Lee, Gibson, Katz, & Patterson (1994) mentions a possible problem known as the "write hole" that affects traditional RAID techniques. Writing operations on RAID array require that data and parity blocks are being written to the disks simultaneously. However, writing operation to multiple independent disks lacks write atomicity. Power or disk failure during writing operation may lead to a situation when the data and parity blocks do not match. If case data is incorrectly written, then in many cases they can be fixed or at least detected by a tool for checking the consistency of a traditional file system (e.g. fsck, CHKDSK) on top of RAID array (www.FreeRaidRecovery.com, 2011, p. 11). In the light of the above, Schmidt (2006) described journaling as an essential property of file systems or storage systems for high availability. By storing a list of pending changes not yet committed back to stable storage, journaling enables maintaining the integrity of the file system and allows fast file system recovery after a crash, "*making them very attractive for systems with high availability requirements*" (Kerrisk, 2010, pp. 260-261). Nevertheless, according to results of Bairavasundaram, Goodson, & Schroeder (2008) study based on observation of the total sample of 1.53 million disk drives over a period of 41 months, averagely, 1 of 90 disks suffers from a silent data corruption, also known as data decay or data rot, resulting in checksum mismatch, lost or misdirected writes, and parity inconsistency. Unfortunately, traditional file systems are unable to detect such corruptions (Prabhakaran, et al., 2005). Yet "next-generation" file systems are able to cope with this problem by using "copy-on-write" technique to provide atomicity for write operations, as well as checksum algorithms for metadata and data (e.g. ZFS, Btrfs, ReFS[13]) combined with self-healing algorithms to detect and eliminate such data corruptions (Salter, 2014).

[13] https://technet.microsoft.com/en-us/library/hh831724.aspx – Resilient File System Overview

3.1.3 Network Availability

High availability systems design also requires highly available network communications (Oggerino, 2001, pp. 5-16). According to Nadeau & Gray (2013, p. 34) there are two major mechanisms to achieve network high availability: *redundancy at the network level* and *redundancy at the element level*. Redundancy at the network level is based on redundant communication paths along with using redundant network equipment and redundant paths in the network design. Redundancy at the element level implies using redundant route processors, switch control modules, and power supplies.

Nevertheless, increasing network redundancy does not necessarily equate to increasing its availability, as growing complexity may actually decrease availability. (Berkowitz, 2002, p. 344)

By adopting TCP/IP model (IETF, 1989) network availability can be built at different levels of the network hierarchy:

- *Link layer* introduces such fault-detection and protection mechanisms as Link Fault Management (LFM) (Sonderegger, Blomberg, Milne, & Palislamovic, 2009, p. 316) to monitor link operation and Spanning Tree Protocol (STP) (Perlman, 1985) as well as its extensions to build a logical loop-free network topology. Moreover, aggregation of multiple network connections (e.g. Link Aggregation Control Protocol (LACP)) to increase both, redundancy and data throughput, can be implemented at this layer (van Vugt, 2014, pp. 27-29).
- *Internet layer* provides an additional level of protection by means of various routing protocols (e.g. Open Shortest Path First (OSPF), Border Gateway Protocol (BGP), etc.). They can be deployed to use multiple alternative network paths (multipathing) or to select the best path through a network, and in case of its unreachability, switch to an alternative path (multihoming). To provide the availability of a shared IP address (e.g. for router or firewall) in case of failover, first hop redundancy protocols (e.g. Virtual Router Redundancy Protocol (VRRP), and Common Address Redundancy Protocol (CARP)) were designed. (Kaur & Gurm, 2015)

  HA clusters use a similar concept of “floating” IP address known as *cluster IP address*. A service provided by HA cluster is accessed using dedicated cluster IP address, which is assigned as alias IP address to the node that executes current service workload. In case of its outage cluster IP address is assigned automatically to other node that takes over the service workload. (Marcus & Stern, 2003, pp. 378-379)

  Additionally, RFC 4786 suggests to use anycast technique to increase availability of network services by announcing the same destination IP address within the scope of an autonomous system or the global internet. Such practice is perfectly suitable for services based on stateless protocols where single request and single reply are small enough to fit one IP packet (e.g. DNS over UDP). (IETF, 2006)

- Availability means at *transport layer* and *application layer* encompass layer-4 and layer-7 load balancers (e.g. HAProxy[14]) to eliminate a single point of failure (SPoF). (Xu, 2005, pp. 18-21)

3.1.4 Clustering

According to Buyya (1999) *clusters* are defined as follows:

> *A cluster is a type of parallel or distributed processing system, which consists of a collection of interconnected stand-alone computers working together as a single, integrated computing resource. (Buyya, 1999, p. 9)*

Initially, clusters were aimed at high-performance computing (HPC) (Hwang, Dongarra, & Fox, 2011, p. 66), and these two terms were interchangeable (Sloan, 2005, p. 11). Currently, under distributed computing paradigm the following types of clusters, depending on their purpose, can be identified (Kahanwal & Singh, 2012):

- *High-availability (HA) clusters* that provide continued service when a system component fails.
- *Load-balancing clusters* that handle a large volume of requests by distributing them across multiple servers.
- *High performance computing clusters* that increase computing throughput by scheduling and executing multiple jobs on separate cluster nodes.

A more comprehensive description of HA clusters was made by Critchley (2015):

> *HA clusters provide continuous availability of services by eliminating SPoFs and by failing over services from one cluster node to another in case a node becomes inoperative. Typically, services in a HA cluster read and write data (via read-write mounted file systems). Therefore, a HA cluster must maintain data integrity as one cluster node takes over control of a service from another cluster node. This is done with volume and lock managers which are integral parts of clustering software. (Critchley, 2015, p. 74)*

HA clustering is considered as one of the best and most versatile solutions to increase the availability of IT systems (IDC, 2013b), that can solve at least the following problems (Shivakumar, 2014, p. 60):

- ensuring maximum availability for any round-the-clock service, regardless of any failures, caused by operating system, data storage, applications, or infrastructure;
- ensure business continuity in case of major disruption;
- improving SLA support, thus customer's effectiveness and satisfaction;
- giving competitive advantage to a business through maximum availability of software and services;

[14] http://www.haproxy.org/ – The Reliable, High Performance TCP/HTTP Load Balancer

- reducing RPO and RTO close to zero for business-critical systems and data;
- adhering to compliance regulations.

Historically, the approach for HA clustering was used in proprietary computing environments (Gartner, 2009; Johnson, 1992; Kronenberg, Levy, & Strecker, 1986). However, some attempts were initiated to build an HA cluster architecture for carrier-grade and mission-critical systems using COTS ecosystem based on open specifications. Service Availability Forum (2011) standardized the interfaces of HA middleware to foster the implementation of high available systems and services built with COTS components. The forum's specification includes Hardware Platform Interface (HPI), which abstracts the hardware from the service availability middleware, and Application Interface Specification (AIS), which abstracts HA middleware and service applications. Open-source realization of this specification is being developed under the OpenSAF[15] project (Toeroe & Tam, 2012, pp. 355-368). In its turn, Carrier Grade Linux Working Group from Linux Foundation[16] incorporates and presents an open architecture for carrier-grade services on Linux kernel-based operating systems (Linux Foundation, 2011). Furthermore, there are several FOSS ready-to-use solutions that provide automatic failure detection, recovery and cluster resource management services (Liebel, 2013, pp. 240-249; Perkov, Pavković, & Petrović, 2011; Schwemmer & Neufeld, 2009).

HA cluster as a distributed system faces the physical constraints displayed in Table 3-3:

Physical Constraint	Tendency	Effect
number of nodes	increasing the probability of failure in a system	reduced availability and increased administrative costs
	increasing the necessity for communication between nodes	reduced performance as scale increases
distance between nodes	increasing the minimum latency (fixed by the speed of light) for communication between distant nodes	reduced performance for certain operations

Table 3-3: Physical constraints and their tendencies in distributed systems, based on (Takada, 2013)

According to Fox & Brewer (1999) any distributed system can possess at most two of the following three properties at the same time:

- Consistency *(C)*
- Availability *(A)*
- Partition tolerance *(P)*

[15] http://www.opensaf.org/ – The Open Service Availability Framework

[16] http://www.linuxfoundation.org/collaborate/workgroups/cgl – The Linux Foundation Carrier Grade Linux Workgroup

This principle was formalized by Gilbert & Lynch (2002) and is known in theoretical computer science as *CAP theorem*. Their formal model of synchronous and asynchronous distributed computing illustrates proof of CAP theorem when there is no synchronization between the nodes of a distributed system (i.e. $P \Rightarrow \neg(C \wedge \mathrm{A})$) and shows feasibility of achieving a practical compromise between consistency and availability for partially synchronous systems (Gilbert & Lynch, 2002).

In context of CAP theorem term *consistency* actually refers to linearizability – also known as atomicity or strong consistency – where data is always the same on all nodes at any given time (in the A*C*ID sense consistency, in simplistic terms, means absence of corruption), *availability* is an ability to return a valid response to any request by any non-failed node in the system even if that node is partitioned off, and the *partition tolerance* implies that splitting into several isolated network partitions does not lead to invalid response from any node. (Gilbert & Lynch, 2002)

According to the CAP theorem, there are only three possible approaches to build a real-world distributed system. Therefore, system designers have to choose two (CA, CP or AP) of three desirable properties depending on the application requirements (i.e. ACID or BASE semantics) and business needs (Shivakumar, 2014, p. 74). Figure 3-1 visualizes such CAP theorem classification by three different intersections:

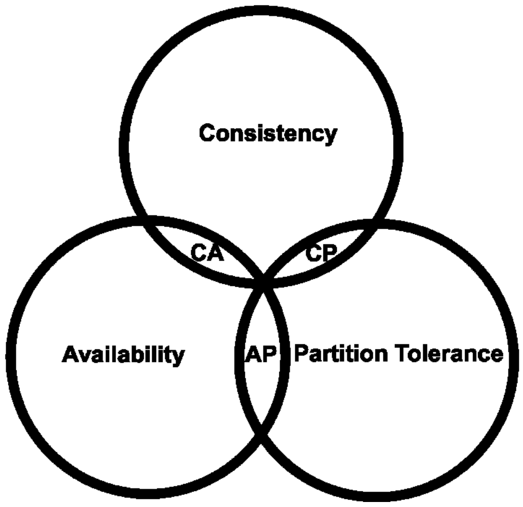


Figure 3-1: Brewer's CAP theorem, based on (Murugesan & Bojanova, 2016, p. 553)

Hale (2010) and Robinson (2010) questioned the possibility to build an applicable to a real-world distributed CA system. They stated that *partition tolerance* is mandatory property for all distributed systems, since network partition is considered as an inevitable fault:

> *For a distributed (i.e., multi-node) system to not require partition-tolerance it would have to run on a network which is guaranteed to never drop messages (or even deliver them late) and whose nodes are guaranteed to never die. (Hale, 2010)*

Therefore, only one trade-off between consistency and availability can exist *when* there is a network partition and *if* there is no network partition.

In case of network partition split cluster systems, which follow CA approach and use data replication may face a problem of a so-called, "*split-brain*" scenario in which different partitions create conflicting replicas. (Obasi, Asagba, & Silas, 2015)

To avoid potential partition split redundant *heartbeat network* with multiple communication paths between all nodes is an essential requirement for an HA cluster. *Heartbeat messaging* is used to check the health of all cluster nodes which are considered to be in failed state when they stop sending periodic “heartbeats” to other nodes in the cluster. In such a situation an immediate cluster reconfiguration is required. Remaining cluster nodes must agree on cluster reconfiguration. To do so, a *quorum*, the minimum number of running nodes in a cluster is required. In case of *majority quorum*, more than half of the cluster nodes should be online to perform “safe” cluster operations. Such an approach prevents unsafe “split-brain” situations when a cluster is split into partitions. Although, a quorum in case of a two-node cluster can only be achieved if both nodes are online. So, if one node fails the other node will not have quorum to perform the cluster operations. Therefore, a quorum approach makes no sense for two-node HA cluster. (Resman, 2015, p. 15)

To prevent a “split brain” scenario in a two-node cluster without quorum, a concept of node *fencing* can be used. The underlying idea of fencing is to withdraw a failed node from all cluster activities. If a cluster node stops to respond the other operating nodes can power it off by using fencing device (e.g. managed power switch, integrated management board, etc.) (van Vugt, 2014, pp. 4-7).

Another fundamental concept for HA clustering, as part of distributed system, is *virtual synchrony* model that is used to coordinate actions across all nodes in clusters, which provides a reliable approach for a strictly ordered message delivery between processes within a process group. Therefore, all operations in clusters are guaranteed to be performed in the same order on all nodes. (Kenneth, 1987)

HA clusters are designed using either *shared-disk* or *shared-nothing* architectural approaches. (Critchley, 2015, p. 338)

Clusters that use shared-disk architecture, as shown in Figure 3-2, each cluster node shares a common set of disks, while having its own memory. The advantage of using a shared-disk architecture for the clusters is in its adaptable performance. Increase of cluster performance can be done either by adding more processors and memory in each node of the cluster, or by adding additional node. Disadvantages of such an approach include complexity and a lack of scalability, as it requires introduction of a distributed locking mechanism and a two-phase commit protocol (2PC) to access the shared data. (Ray, 2009, pp. 23-24)

In the clusters based on shared-nothing architecture, which is depicted in Figure 3-3, each node has its own memory and its own disk, which are not shared by the other nodes of the cluster. In such an architecture, only a single interconnection network is shared between the nodes of the cluster. Shared-nothing architecture provides better scalability, extensibility and availability for the clusters. A much higher cost of communication for accessing a non-local disk is seen as a main drawback of shared-nothing architecture. (Ray, 2009, pp. 24-25)

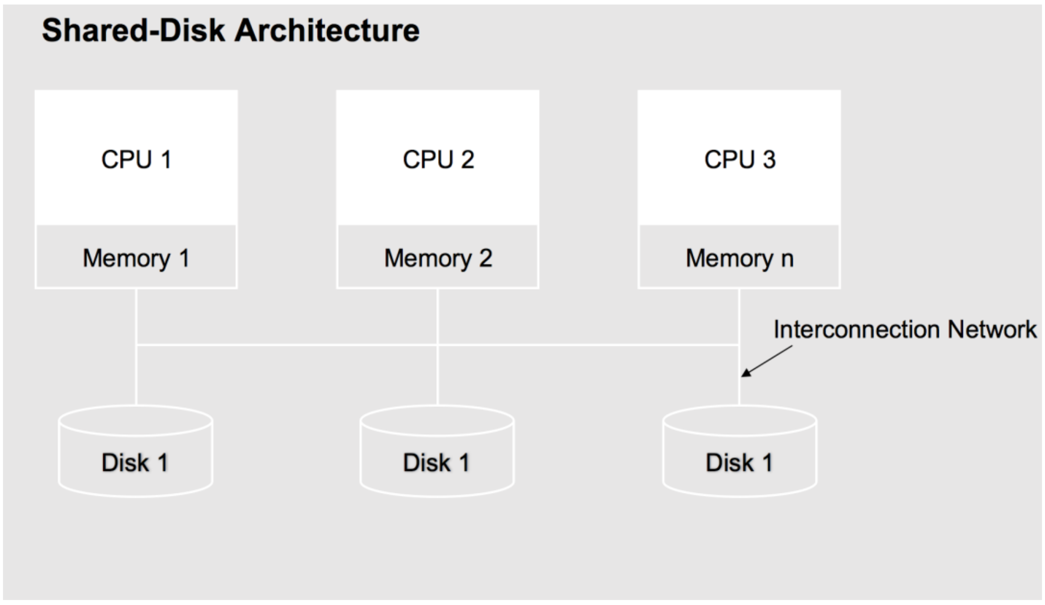


Figure 3-2: Shared-disk architecture, based on (Mullins, 2002, p. 58)

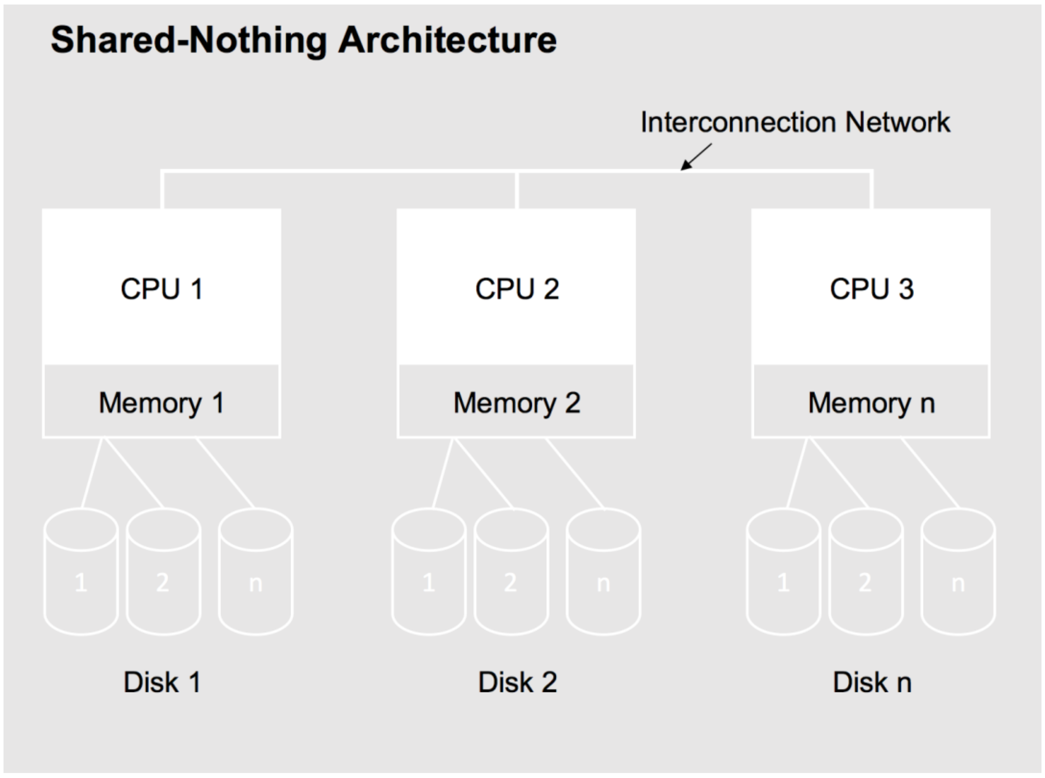


Figure 3-3: Shared-nothing architecture, based on (Mullins, 2002, p. 59)

3.1.5 Data Replication

The availability of a system could be increased by means of storing the same data in multiple storage devices, a process known as *data replication*. (Bernstein, Hadzilacos, & Goodman, 1987, p. 265)

There are two types of replication, as shown in Figure 3-4, that can be distinguished: *synchronous replication*, and *asynchronous replication.* Synchronous replication implies that I/O write operation has to be written and confirmed by both replicas, primary and secondary, before an application can proceed. In its turn, during asynchronous replication, I/O completion confirmation is only receive from the primary replica, allowing application to proceed without waiting for additional confirmation from the secondary replica. (Orenstein, 2003, p. 73)

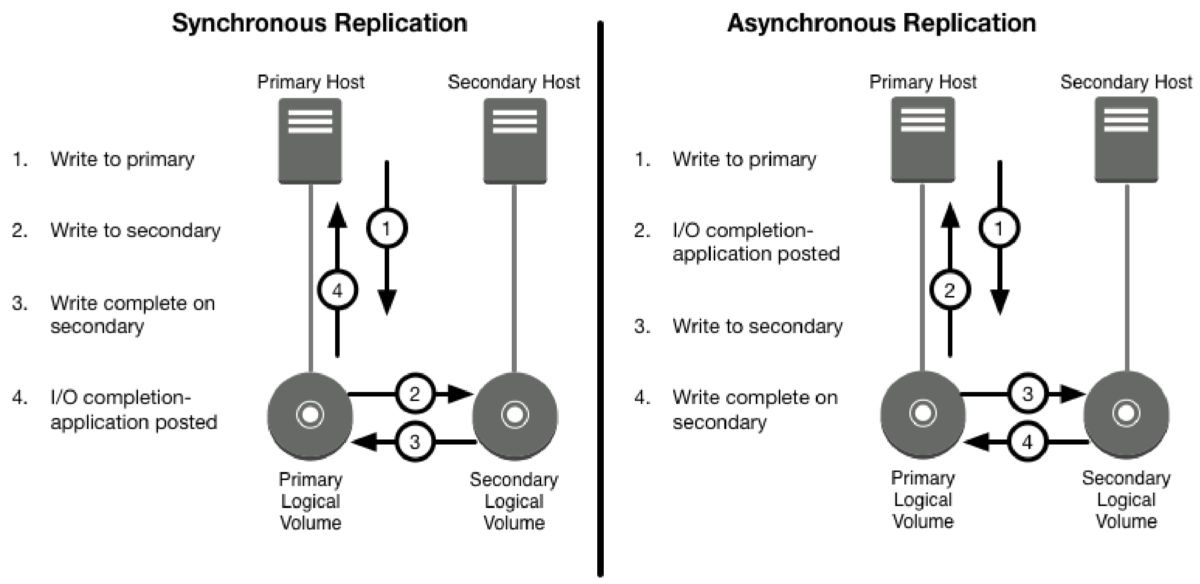


Figure 3-4: Synchronous and asynchronous replication, based on (Orenstein, 2003, p. 74)

The choice of replication type depends on RTOs and RPOs (Allspaw & Robbins, 2010, pp. 287-288). For example, to ensure "zero data loss" synchronous replication can be a feasible option (Critchley, 2015, p. 360). However, synchronous replication can also negatively affect systems performance by introducing delays, making the slowest replica a bottleneck, as most applications wait for I/O confirmation (Orenstein, 2003, p. 73). Thus, "best practices" suggest using asynchronous synchronization if round-trip time (RTT) of replication links more than five milliseconds (ms) (Oracle, 2015a, p. 9; Tate, et al., 2013, p. 182). For reference, it takes about 1 ms for light to travel 100 km out and back through the glass core of the fiber (cf. (Miller, 2012)), therefore this RTT should be considered as minimal latency for each atomic write I/O operation between two replicas some 100 km away.

Database replication can use a transactional approach, in which all changes within *transactions*, have to be committed. Such an approach guarantees the same *ACID* (Atomicity, Consistency, Isolation, Durability) properties for each distributed transaction between the original and replicated databases (Bernadette Charron-Bost, 2010, pp. 219-220). The consistency property is ensured by using atomic commitment protocols (e.g. 2PC) combined with redo and write-ahead protocols, where a distributed transaction is not committed until it is executed at all replicas (Rob, Coronel, & Crockett, 2008, pp. 686-687). ACID properties are inseparable, so discarding any of them makes the rest of combination meaningless (Härder & Reuter, 1983).

3.1.6 Virtualization

The growth of a server virtualization and infrastructure virtualization provides new opportunities for an end-to-end application availability (Gartner, 2009). However, unlike common misconception, "virtualizing the application environment doesn't mean that it automatically becomes resilient" (Heavy Reading, 2012, p. 5).

The virtualization concept has its roots from attempt to partition a modified IBM System/360 Model 40 mainframe computer running CP-40 operating system into *virtual machines* during the mid-1960s (Adair, Bayles, Comeau, & Creasy, 1966; IBM, 2012). A virtual machine (VM) is managed by a *hypervisor* or a *virtual machine monitor* (VMM), which can be either type-1, native, (e.g. VMware ESXi[17], Microsoft Hyper-V[18]), or type-2, a hosted, hypervisor (e.g. VirtualBox[19], QEMU[20]) (Popek & Goldberg, 1974). Moreover, an operating-system-level, or a *container*, virtualization (e.g. Docker[21], LXC[22]) is a widely used approach to mitigate an overhead introduced by a virtualization layer (Joy, 2015). For example, a disk I/O overhead may vary from 7% for container-based (Docker) to more than 59% for type-1 virtualization (KVM[23]) (Morabito, Kjällman, & Komu, 2015).

As alternative to a reactive failover approach to achieve high availability, Bressoud & Schneider (1996) proposed to use a proactive hypervisor-based fault-tolerance technique, when the primary and the backup VMs execute the same set of instructions in a virtual *lock-stepping* fashion using replica-coordination protocols. However, such a deterministic replay in the lockstep mode imposes stricter constraints on the architecture of the target node than a simple virtualization and it cannot be easily extended for the multi-core CPUs (Cully, et al., 2008).

Alternatively, Cully, et al. (2008) elaborated an idea of *continuous check-pointing* the running VM and replicating its externally visible state (i.e. "dirty" memory pages, disk blocks, CPU state) to the backup VM asynchronously at very high frequencies (e.g. every 25 ms). The project called Remus[24] implements the described concept of a continuous live migration of VM from the primary

[17] http://www.vmware.com/products/esxi-and-esx/ – VMware ESXi

[18] http://www.microsoft.com/hyper-v – Microsoft Hyper-V

[19] https://www.virtualbox.org/ – Oracle VM VirtualBox

[20] http://qemu.org/ – Quick Emulator

[21] https://www.docker.com/ – Docker

[22] http://linuxcontainers.org/ – Linux Containers

[23] http://www.linux-kvm.org/ – Kernel-based Virtual Machine

[24] https://wiki.xenproject.org/wiki/Remus

physical host to the backup. It was already merged to the recent versions of open-source hypervisor Xen[25]. Implemented in the Kemari project[26] an adaptive event-triggered approach based on network or disk activity to check-pointing running VM is an alternative to the periodical check-pointing at fixed intervals (Tamura, Sato, Kihara, & Moriai, 2008). A similar concept based on a continuous check-pointing is now adopted by VMware as Fault Tolerance feature for their ESXi hypervisor (VMware, Inc., 2016). However, such check-pointing approaches might suffer from significant performance overhead (Lu & Chiueh, 2009; Zhu, et al., 2010; Sun & Blough, 2010).

Portnoy (2012, p. 14) listed the opportunities for availability which can be achieved with introduction of virtualization:

- VMs can be moved from one physical host to another without interruption
- Additional resources (e.g. CPU, memory, disk storage), can be added “on fly” without the need to reboot guest VM (if it is supported by an operating system in a guest VM)
- Off-site VMs replication for a faster disaster recovery

Heavy Reading[27], an independent research organization, examined the impact of virtualization on availability of application and infrastructure and came to conclusion that virtualization may compound a high availability burden by introducing a complexity, since a new layer needs to be managed and orchestrated. HA needs for applications in virtual environments are depicted in Figure 3-5. Furthermore, various hypervisors provide different maturity levels and high-availability capabilities. (Heavy Reading, 2012)

Yet, Suresh & Kannan (2014) outlined practical problems of virtualization, depicted in Figure 3-5, that have to be solved. The overall fairness and performance are considered as pitfalls of virtualization. In this regard, improving the process and I/O scheduling, better dynamic resource allocation, reducing overall complexity of a VMM as a potential source of security breaches and performance bottlenecks, live migration of VMs between hosts with different processor architectures, deployment of unified management for heterogeneous virtual environments, implementing hardware offload for some memory and I/O management still remain open challenges.

[25] https://wiki.xenproject.org/wiki/Xen_Project_Release_Features#Features

[26] https://www.openhub.net/p/kemari

[27] http://www.heavyreading.com/

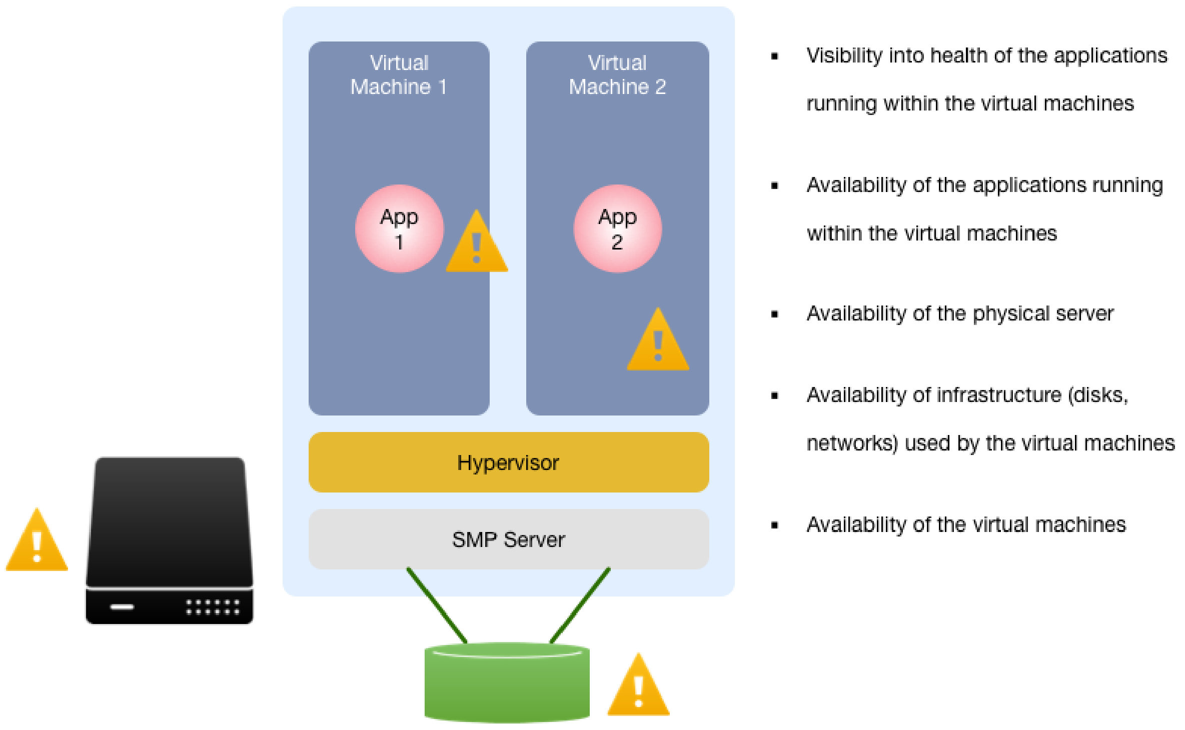

Virtual Machine 1
Virtual Machine 2
App 1
App 2
Hypervisor
SMP Server
Visibility into health of the applications running within the virtual machines
Availability of the applications running within the virtual machines
Availability of the physical server
Availability of infrastructure (disks, networks) used by the virtual machines
Availability of the virtual machines

Figure 3-5: HA needs for applications in virtual environments, based on (Heavy Reading, 2012, p. 11)

3.1.7 Software-Defined Anything

The cutting edge Software-defined anything (SDx) concept is also significant in context of building resilient IT environments. This concept implies the transfer of key IT infrastructure functions to the software level, enabling its scalability, manageability, reliability, and interoperability with other parts. Studies conducted by Gartner (2013) and NEC (2016) list Software Defined Anything within modern key trends. In fact, it is the further extension of virtualization vision (Zhu, Song, Ni, & Ren, 2016, p. 99), where a software-configurable platform (e.g. network equipment, storage systems, load balancers, etc.), functions as a virtual machine on standard servers. SDx is an umbrella concept under which such approaches as *software-defined networking* (SDN) and *software-defined storage* (SDS) be can be already placed.

Heegaard (2015) discussed the application of an SDN approach for achieving dependability, when the entire network intelligence is off-loaded to specialized applications running on a dedicated device, called SDN controller. SDN controller (control plane) manages traffic flows by sending instructions using OpenFlow[28] to switches (data plane) to perform various actions on traffic flows (e.g. allow, deny, redirect, rewrite header fields in packets, etc.). Such ideology of decoupling control and data planes within a single device allows to build cheaper less-intelligent network devices (Erel, Arslan, Yusuf, & Canberk, 2015, p. 754). However, such approach of centrally managed network turns a single instance of SDN controller into a single point of failure (Nadeau & Gray, 2013, p. 39).

[28] https://www.opennetworking.org/sdn-resources/openflow/ – OpenFlow project website

In its turn, software-defined storages is a data storage abstraction over physical storage hardware (Storage Networking Industry Association, 2015). Like SDN, SDS encompasses both data and control paths (Carlson, Yoder, Schoeb, Deel, & Pratt, 2014). However, in contrast to SDN, SDS has an essential difference, which makes its approach more complicated. Network element within SDN just receives and transmits data following instructions provided by SDN controllers, but in SDS "less intelligent" storage devices should also store the related data.

SDS is commonly implemented as a management server (or a cluster) that abstracts and aggregates different types of underlying physical storage from various vendors with different interfaces and connection protocols. (Schulz, 2017, pp. 9-14)

Recent researches in SDS conducted by Ohtsuji & Tatebe (2015) shows new perspectives for replication over the network by using modern Ethernet technologies combined with remote direct memory access (RDMA) mechanism by minimizing the number of memory copy operations over network. RDMA tries to overcome the overhead introduced by the TCP/IP stack of operating systems and to offload the system's main CPU for other tasks. Such software realizations have already emerged and demonstrated their efficiency.

3.1.8 Disaster Recovery

The disaster recovery (DR) is considered as extension of the high availability (Lumpp, et al., 2008). Business Continuity Institute[29] defined the disaster recovery as "*strategies and plans for recovering and restoring the organizations technological infrastructure and capabilities after a serious interruption*" (Business Continuity Institute, 2011, p. 20).

Figure 3-6 outlines the major causes for disaster declarations, based on data from research conducted by Sungard Availability Services[30] (Sungard Availability Services, 2014, p. 4):

[29] http://www.thebci.org/ – The Business Continuity Institute (BCI) is the world's leading institute for business continuity.

[30] http://www.sungardas.com/ – Sungard Availability Services

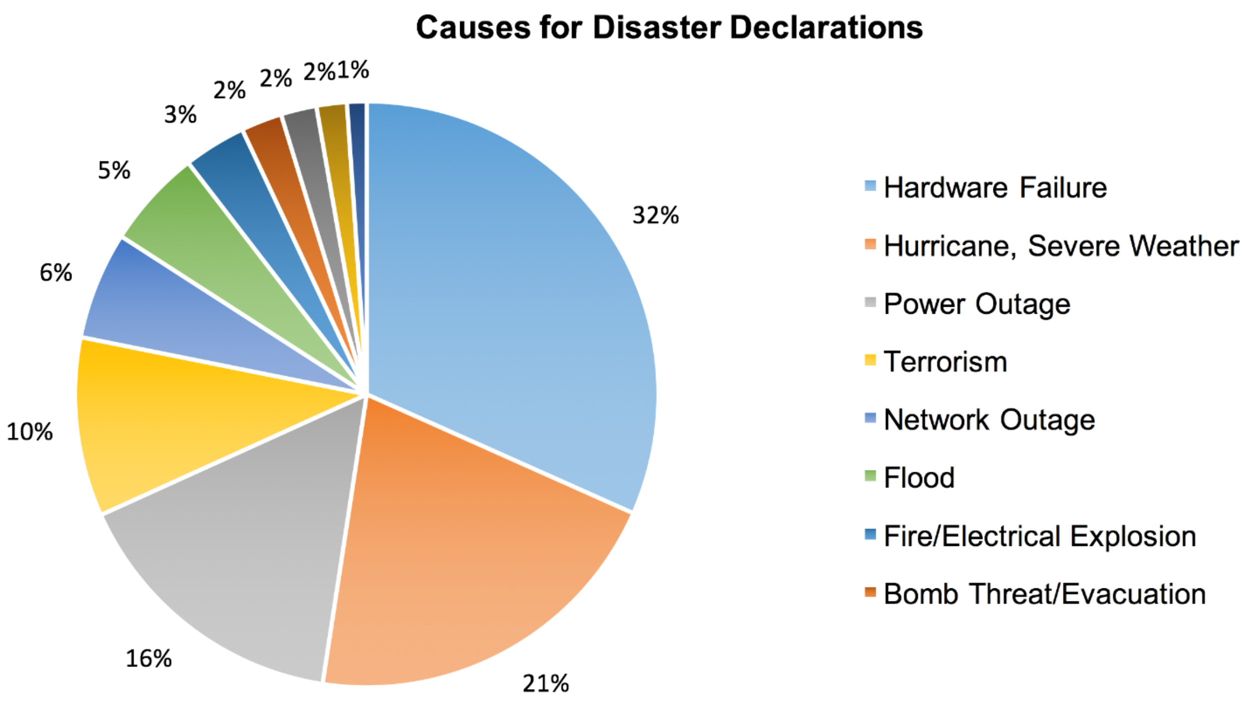


Figure 3-6: Causes for disaster declarations, based on (Sungard Availability Services, 2014, p. 4)

The technical aspect of DR strategy within a business continuity planning involves the process of creating and maintaining a backup site (i.e. data center), performing routine backups and store them at alternate locations, hardening and protecting hardware from environmental damage, eliminating power-related problems, etc. (Hawkins, Yen, & Chou, 2000). A report by Disaster Recovery Preparedness Council (2014) shows that the deployment of a backup site is the most widely used implementation strategy for DR. Such a solution allows mitigating the impact of a disaster or a major business disruption by using a replication for mission-critical and business-critical data and applications between geographically separated sites (Disaster Recovery Journal, 2014). Thus, a backup site has the latest copy of the data and makes it possible to restore the business process in a timely manner. According to Li X. (2012) a new cloud service model "Disaster Recovery as a Service" (DRaaS) can be a feasible low-cost alternative to a dedicated backup site infrastructure.

SHARE user group[31] classified common strategies of enterprise systems to achieve the shrinking RTOs and RPOs and encapsulated them into seven tiers of DR, as shown in Table 3-4 (Bauer, Randee, & Eustace, 2011, p. 18):

DR Tier	DR Strategy
Tier 0: No Off - Site Data	Tier 0 enterprises have no disaster recovery plan and no saved data. Recovery time from disaster may take weeks or longer and may ultimately be unsuccessful.

[31] http://www.share.org/ – SHARE is an independent volunteer-run information technology association

Tier 1: Data Backup with No Hot Site	Tier 1 enterprises maintain data backups offsite but do not maintain a hot site. Backup data must typically be physically retrieved (so - called pickup truck access method, PTAM), and thus significant time is required to access backup media. Since Tier 1 enterprises may not maintain their own redundant servers to recover service onto, time may be required to locate and configure appropriate systems.
Tier 2: Data Backup with a Hot Site	Tier 2 enterprises maintain data backups as well as a hot site, and thus recovery times are faster and more predictable than in Tier 1.
Tier 3: Electronic Vaulting	Tier 3 enterprises maintain critical data in an electronic vault so that backup data is network accessible to the hot site rather than requiring backup media to be physically retrieved and transported to the hot site.
Tier 4: Point-in-Time Copies	Tier 4 enterprises maintain more timely point-in-time backups of critical data so that more timely backup data is network accessible to the hot site, thus reducing the RPO.
Tier 5: Transaction Integrity	Tier 5 enterprises assure that transactions are consistent between production systems and recovery sites. Thus, there should be little or no data loss from a disaster.
Tier 6: Zero or Little Data Loss	Tier 6 enterprises have little or no tolerance for data loss and thus must maintain the highest level of data consistency between production and recovery sites. Techniques like disk mirroring and synchronous I/O are generally deployed by Tier 6 enterprises to minimize RPO.
Tier 7: Highly Automated, Business-Integrated Solution	Tier 7 enterprises automate disaster recovery of Tier 6 enterprises, thus shortening the RTO and with minimal RPO.

Table 3-4: Seven tiers of DR, based on (Bauer, Randee, & Eustace, 2011, pp. 18-19)

3.2 Architectural Patterns for High Availability

This section presents architectural patterns to attain high availability in form of best practices, which were captured during the process of a literature review.

3.2.1 No Single Point of Failure

The most important approach in high availability design is an elimination of "*single point of failure*" (SPoF) – the weakest link in the chain of availability. (Marcus & Stern, 2003, p. 78)

An architecture where the entire system stops from working when one of its components fails should be avoided. It can achieved by introducing *redundancy* at all levels of a system, from redundant network connections and power supplies to redundant array of independent disks (RAID). (Laan, 2017, pp. 64-65)

The simplest way to achieve a redundancy of some component is to duplicate it. If one component fails, the "backup" component takes over the operations of the "partner". This mode of operation is called *active/standby* redundancy. The process of taking control from failing component to the backup component is known as a *failover* or, in some cases as a *switchover*. (Critchley, 2015, pp. 57-58)

3.2.2 Cluster Configurations

A computer cluster (see details in 3.1.4) can have different configuration. Two-node cluster is the minimum required configuration to achieve high availability (Lehmann, 2009, p. 78). But often clusters contain more nodes. All these configurations can be described by one of the following models, listed in Table 3-5 (Resman, 2015, pp. 5-6):

Model	Description
Active/Active	The Active/Active cluster configuration can be used with two or more cluster members. The service provided by the cluster is simultaneously active on all cluster nodes at any given time. The traffic can be passed to any of the existing cluster nodes if a suitable load balancing solution has been implemented. If no load balancing solution has been implemented, the Active/Active configuration can be used to reduce the time it takes to fail over applications and services from the failed cluster node.
Active/Passive	The Active/Passive cluster configuration can be used with two or more cluster members. At a given time, the service is provided only by the current master cluster node. If the master node fails, automatic reconfiguration of the cluster is triggered and the traffic is switched to one of the operational cluster nodes.
N + 1	The N over 1 cluster configuration can be used with two or more cluster members. If only two cluster members are available, the configuration degenerates to the Active/Passive configuration. The N over 1 configuration implies the presence of N cluster members in an active/active configuration with one cluster member in backup or hot standby. The standby cluster member is ready to take over any of the failed cluster node responsibilities at any given time.

N + M	The N over M cluster configuration can only be used with more than two cluster members. This configuration is an upgrade of the N over 1 cluster configuration where N cluster members are in Active/Active state and M cluster members are in backup or hot standby mode. This is often used in situations where active cluster members manage many services and two or more backup cluster members are required to fulfill the cluster failover requirements.
N-to-1	The N-to-1 cluster configuration is similar to the N over 1 configuration and can be used with two or more cluster members. If there are only two cluster nodes, this configuration degenerates to Active/Passive. In the N-to-1 configuration, the backup or hot standby cluster member becomes temporarily active for the time period of failed cluster node recovery. When the failed cluster node is recovered, services are failed over to the original cluster node.
N-to-N	The N-to-N cluster configuration is similar to the N over M configuration and can be used with more than two cluster nodes. This configuration is an upgrade of the N-to-1 configuration and is used in situations where the need for extra redundancy is required on all active nodes.

Table 3-5: HA cluster configurations (Resman, 2015, pp. 5-6)

The selection of a cluster configuration is based on application requirements in normal operations and also in degraded mode (Critchley, 2015, pp. 338-337).

For example, an active/active configuration may offer both scalability and high availability by running one application on multiple nodes behind load balancing, as seen in Figure 3-7. Load balancer distributed the workload between the cluster nodes and may use appropriate mechanism to detect a failed node (e.g. “heartbeats’) and exclude it from cluster workload (Shivakumar, 2014). However, in this case load balancer itself can be described as single of point of failure (see details in Subsection 3.2.1). Therefore, load balancers itself can be configured as active/passive HA cluster, as shown in Figure 3-8, to be prone of outages.

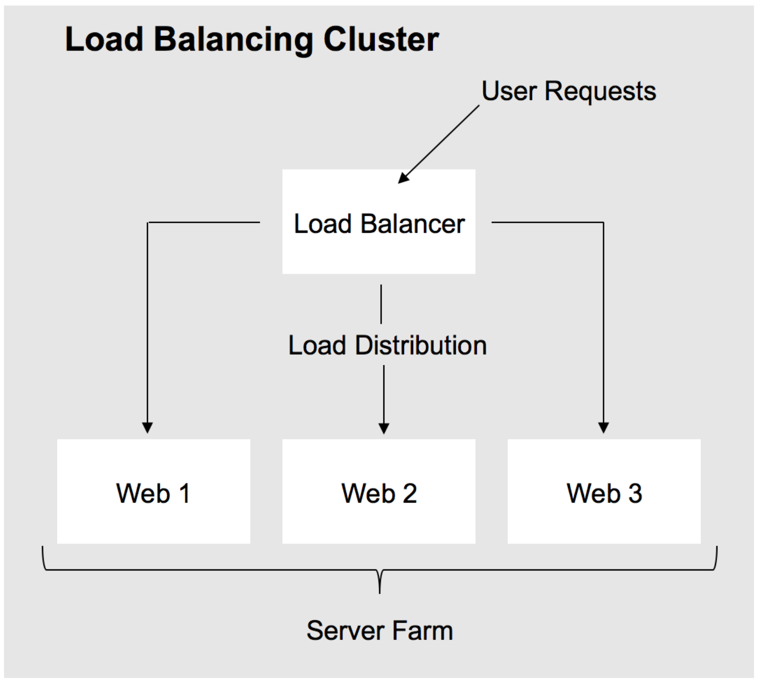


Figure 3-7: Overview of load balancing clusters, based on (van Vugt, 2014, p. 2)

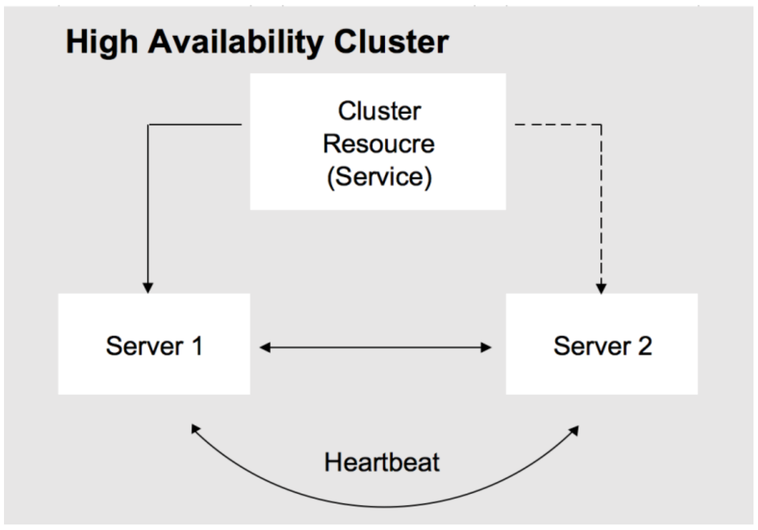


Figure 3-8: Overview of high availability clusters, based on (van Vugt, 2014, p. 3)

3.2.3 Simplicity

A number of authors describe simplicity as a tenet of availability. The more complex a system, the less stable it is, the more potential points of failure it has, and the more difficult it is to manage (Atchison, 2016, p. 63); (Laan, 2017, p. 63); (Piedad & Hawkins, 2001, p. 34). Therefore, unnecessary complicity can be considered as another approach for HA system design.

According to Marcus & Stern (2003) the following consideration should be undertaken when introduce simplicity into the complex systems (Marcus & Stern, 2003, pp. 101-103, 270):

- Eliminate of extraneous hardware on critical systems.
- Run only essential applications on productive systems.
- Disconnect servers from networks they do not need to be.
- Use the names for system (i.e. hostnames) that are easy to remember and pronounce.
- Automate routine tasks to reduce the chances of human error.
- Remove ambiguity from the environment.
- Reducing the numbers of system vendors and models.

3.2.4 Multi-Layered Approach

Modern IT systems and their agile environments are composed of many heterogeneous components and their complexity is constantly growing. To manage such complexity and to maintain decoupling of concerns in system and software engineering an abstraction technique is commonly used. A layered approach for high availability based on contemporary architectural paradigm is a commonly adopted practice. (Slåtten, Herrmann, & Kraemer, 2012, p. 144)

Within the enterprise IT landscape, The Open Group (2011, pp. 491-522) has tried to formalize and to outline reference IT architecture based on the following three major pillars: application, application platform, and communications infrastructure, as seen in Figure 3-9:

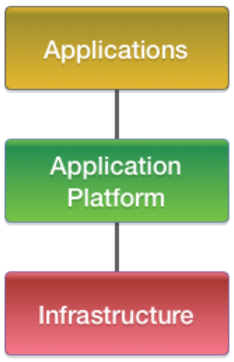


Figure 3-9: TOGAF Technical Reference Model, based on (Harrison, 2013, p. 180)

Additionally HA Forum (2001, pp. 30-32) proposes an outlined model to describe a framework for open architecture high availability systems, which includes:

- platform hardware
- operating system, along with two kinds of middleware are defined: HA management middleware and other middleware
- application

The Figure 3-10 depicts such system model based on the Open HA Framework:

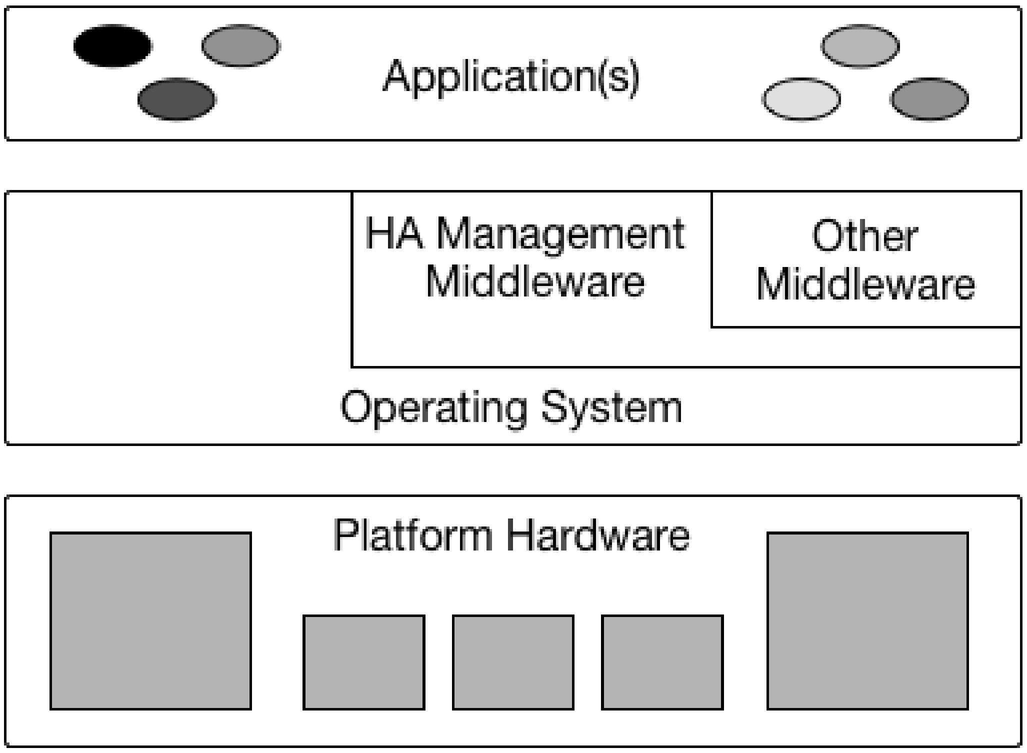


Figure 3-10: Open HA Framework Individual System Model, based on (HA Forum, 2001, p. 31)

Service Availability Forum (2011, p. 25) slightly refines this basic architecture by gluing components with application and hardware platform interfaces. In their study, (Trinitis & Walter, 2003) evaluate a layered approach and demonstrate its advantage over traditional monolithic approaches for a complete balanced high availability base through all components of a system. In the context of virtualized environments, Marshall & Lowe (2015, p. 377) suggest decoupling physical and virtualization layers of high availability, while leaving the other two layers untouched.

4 HIGH AVAILABILITY FOR LOGISTICS SYSTEMS

"A complex system that works is invariably found to have evolved from a simple system that works. The inverse proposition also appears to be true: A complex system designed from scratch never works and cannot be made to work."
by John Gall

This section provides a description of a real-world situation and the general approach that was taken to deal with the research question underling this thesis. First, a use case scenario for a logistics system was introduced to set high availability within the context of logistics systems. The most common failures affecting the logistics system under investigation was examined. Based on the provided objectives and requirements for an HA solution, reference HA architecture in the form of a two-node shared-nothing failover cluster was outlined to create a prototype for the experimental setup.

4.1 Use Case Scenario

This section describes an observation of a real-world situation in the form of a use-case scenario which was used for the process of HA architecture design and subsequent experimental setup.

4.1.1 Warehouse Control System

The logistics system under investigation was a Tier 1 KiSoft warehouse control system ("KiSoft WCS") developed by the KNAPP Company, one of the one of the world market leading suppliers for high-performance logistics solutions (KNAPP, 2016). Following the "zero-defect" warehousing philosophy KNAPP wanted to add HA capabilities to WCS to increase its acceptance by amplified demands of enterprise customers for always-on error-free custom-tailored intralogistics services in the era of "Logistics 4.0" (KNAPP, 2017).

4.1.2 Failure Causes

Statistics based on data from 835 incidents of unplanned interruption of KiSoft WCS reported by customers and documented by the KNAPP service desk in 2015 gave insight into possible failure scenarios. Figure 4-1 presents collected IT-related causes of outages in warehouse operation, grouped in 7 categories:

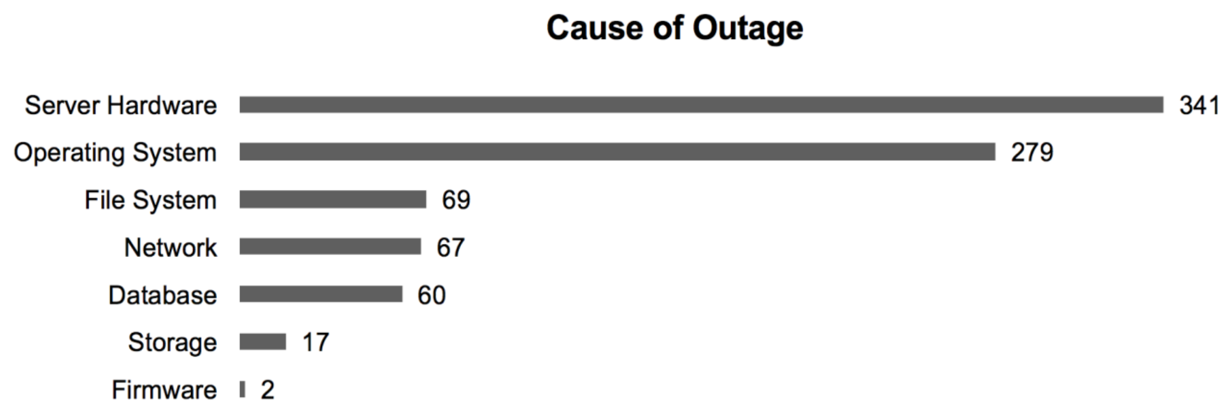


Figure 4-1: IT-related failures that caused unplanned interruption of KiSoft WCS, based on data from (KNAPP, 2015)

The most numerous category titled "*Server Hardware*" includes 341 outages caused by malfunction hardware (e.g. CPU or memory failure, failed system board, power supply or periphery).

The category named as "*Operating System*" contains 279 operating system failures (e.g. system freezes or crashes, dead or runaway processes, out of memory conditions, driver-related problems) that caused interruption in warehouse operation.

The "*File System*" category includes outages caused by file system corruptions or running of space conditions.

The "*Network*" category combined outages introduced by failed network cards or network media.

Outages grouped in the "*Database*" category cover various failures caused by corruption of database files, database content, database log or indexes along with deadlock situations.

Outages in the "*Storage*" category encompasses failures of the storage subsystem (e.g. disk drive or RAID controller). However, it should be noted, that many outages caused by failed disk media were misleadingly assigned to the "Server Hardware" category in incident description.

Finally, there were only two outages caused by firmware failures.

Understanding the causes of outages helped to increase resilience, and when possible, tolerance to such failures in the proposed HA architectural approach.

4.1.3 Objectives and Requirements

In order to implement HA solution for KiSoft WCS which is suitable to the warehouse environment a number of functional and non-functional requirements on the HA architecture were considered.

First, a business impact analysis was conducted, which resulted in the following objectives for HA solution:

- mitigate risk of unplanned outage;
- avoid unplanned outage longer than five minutes;
- prevent data corruption and data loss.

Further, the following requirements for HA solutions were provided:

- The HA solution has to support KiSoft WCS without modification;
- The HA solution has to be installed on-site to ensure the lowest latencies (less than 1 millisecond) to equipment controllers that direct material flow activities on a near real-time basis;
- The HA solution has to tolerate a single hardware failure;
- The HA solution has to be hardware agnostic to support any standard server platform;
- The HA solution has to exclude complexity in its ongoing maintenance;
- The HA solution has to utilize free and open-source software backed by open and "industry-ready" standards to provide affordable and flexible implementation and to prevent proprietary vendor or technology lock-in.

HA objectives, captured in Table 4-1, were used as quantitative targets to evaluate the proposed HA architectural approach by implementing and testing the prototype:

Objective Category	Objective	Property
Data availability	RPO	0
System availability	RTO	< 300 seconds
	MTTF	High
	MTTR	Low
Network availability	RTT	< 1 ms

Table 4-1: HA objectives catalogue for KiSoft WCS

4.2 Outline of HA Architecture

This section presents a reference HA architecture in the form of an HA cluster as a solution, based on requirements from the real-world scenario (as stated in Section 4.1).

4.2.1 Approach

In designing the reference HA architecture the following two criteria were applied:

- soft real-time requirement for the investigated logistics system constrained general solution space to a *locally hosted cluster*
- single-node semantics of the logistics system qualified *failover clustering* approach

The following principles were reflected in the HA architecture:

- Simplicity of integral parts, either of which can be temporarily disabled or replaced
- Use of off-the-shelf solutions, absence of the "most reliable hardware/software"
- Eliminating SPoFs

By combining the TOGAF Technical Reference Model with the system model from the Open HA Framework (for details see Subsection 3.2.4), the proposed HA architecture was organized into the following functional layers, as shown in Figure 4-2:

- An *Infrastructure* layer that encompasses server hardware and related equipment (e.g. server racks, KVM switches, HVAC, etc.), network and electric power systems components.
- A *Platform* layer that includes an operating system and essential execution environment (e.g. file systems, specific shared libraries, environment variables, etc.) for upper layer, as well as "workload-agnostic" mechanism for data replication.
- An *Application* layer made of clustering software, and the logistics system itself along with the required database system and middleware.

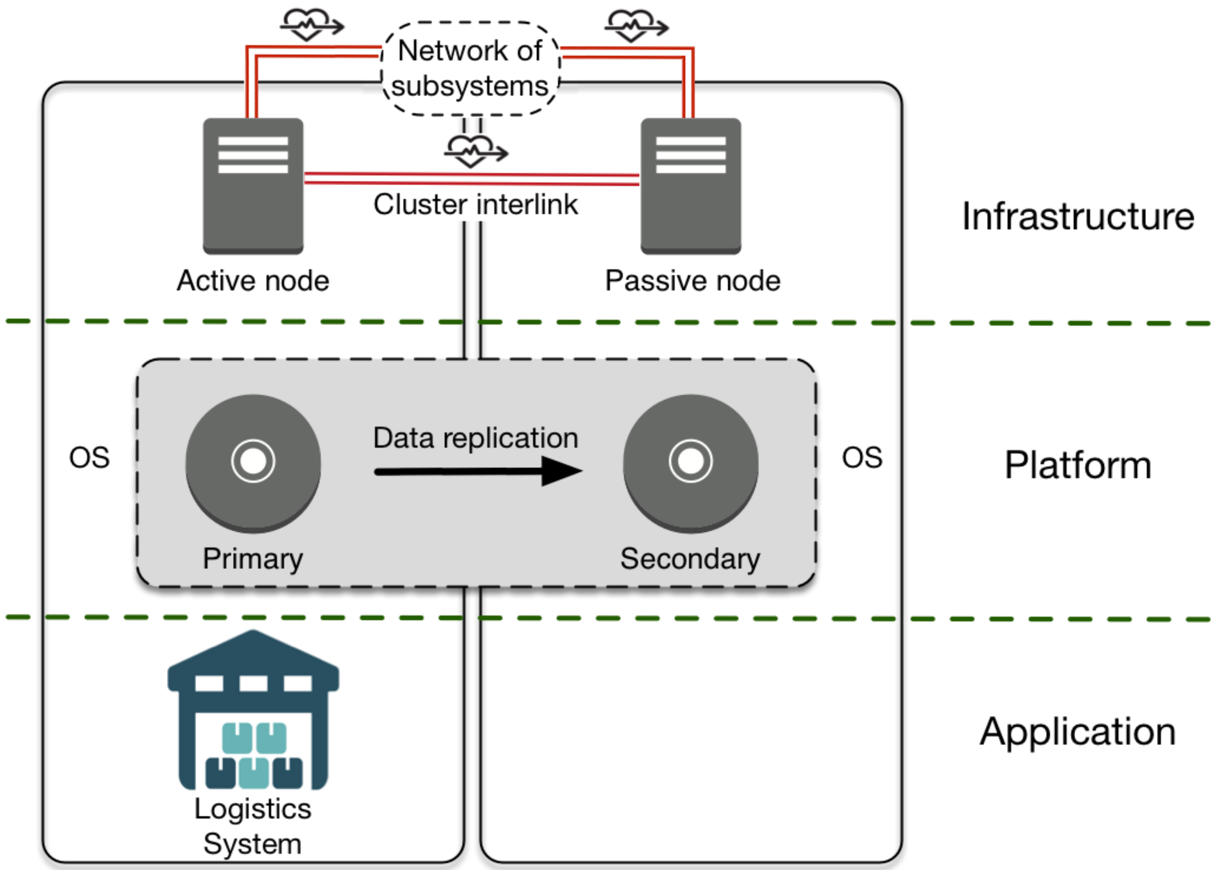


Figure 4-2: Proposed layered HA cluster architecture

4.2.2 Model

To provide minimum required N+1 redundancy the proposed reference architecture encapsulates asymmetric two-node HA cluster system ("twin-node cluster") in accordance with the *active/passive model* (for details see Subsection 3.2.2). Both nodes are running at any moment in time, but only one node in the cluster, hereafter referred to as the *active node*, is supposed to execute the current workload of the logistics system. The other node, hereafter referred to as the *passive node*, does not execute any functional workload except data replication from the active node and maintaining cluster membership functions. Such an approach makes possible to run the logistics system without adjustments by maintaining existing single-node semantics.

Additionally, workload migration can be triggered either manually (*switchover*) or automatically (*failover*):

- Switchover is supposed to be done in case of maintenance activities (replacing malfunction or faulty hardware elements). During the switchover procedure cluster software gracefully stops the logistics system and all related resources on the active node and starts them on the passive node.
- In case of system crash of active node process of failover is trigged automatically by cluster software of the passive node and accordingly the logistics system and other relevant resources are started there. Essentially, such non-transparent failover processes do not assume restoration of a previous program state for the logistics system, but applications have access to exactly the same data that was written to the non-volatile storage of active node, up to the time of system crash. It is assumed that transactional consistency is provided by the database management system to ensure data integrity after such a system crash.

  In case of failed node recovery, *failback*, migration of the logistics system to the original node, is not required.

The proposed architectural approach can deliver AEC-2 “High Availability”-class implementation (for details see Table 2-3) using the simplest possible configuration (for details about approach see Subsection 3.2.3).

4.2.3 Communication

Each node in the proposed HA architecture for logistics system includes at least two independent paths to communicate with each other over the network (for details see Subsection 3.1.3):

- *cluster interlink* has to be deployed as redundant, direct point-to-point (i.e. without using an active network component), high-throughput and low-latency network interconnection between the nodes to ensure flawless synchronous replication of data and cluster messaging (i.e. “heartbeating”)
- *uplink to external network* (i.e. network of subsystems) has to be deployed redundantly to provide uninterruptable communication with external systems and reliable cluster messaging

Each node has unique network settings to avoid IP address conflicts. Network communication of logistics system with any external system (e.g. WCS and WMS or WCS and PLC) is done using a “floating” cluster IP address, which is assigned as a secondary IP address to a network interface of the active node (for details see Subsection 3.1.3). In case of switchover or failover there is no necessity to change any setting on the related systems. It is assumed that such systems can detect network failure and recover from it (cf. Subsection 1.2.2 of (IETF, 1989)) by reestablishing connection to the cluster IP address of the logistics system.

4.2.4 Data Replication

To ensure data protection in the case of hardware or node failure synchronous data replication between the nodes was proposed (for details see Subsection 3.1.5). Only the active node has a read/write access to the replicated data. The passive node just stores the data on the directly attached storage system (i.e. RAID) and prohibits any access to this data until switchover or fail-over has occurred.

Synchronous data replication is supposed to be deployed using software-defined storage approach (for details see Subsection 3.1.7) utilizing commodity hardware (i.e. directly attached to RAID controller hard or solid-state drives) that provides lowest latencies (i.e. RDMA support). Data replication processes utilize the bandwidth of cluster interlink connection.

As there is no central shared storage system (i.e. SAN) application of the proposed approach results in shared-nothing cluster implementation (for details see Subsection 3.1.4) that stores data into two physically different places synchronously.

4.2.5 Redundancy

The proposed HA architecture assumes incorporated redundancy on all layers, which can be extended when necessary to eliminate single points of failure (for details see Subsection 3.2.1):

- *Infrastructure layer*: multiple network paths, twin network adapters, redundant power suppliers, RAID arrays with spare drives, fans, etc.
- *Platform layer*: data replication, journaling file system, alternative network routes, etc.
- *Application layer*: database log shipping, various error detection and correction methods implemented in software, use of multiple NTP and DNS servers, etc.

4.2.6 Quorum

In the two-node cluster, a quorum can be obtained only if both nodes are online (for details see Subsection 3.1.4). Hence, a process of failover cannot be triggered if there is only one node left cluster. Therefore, the proposed approach assumes that the quorum option in cluster software should not be used.

To prevent problems in degraded two-node clusters (i.e. only one node is in known state) without quorum, use of the fencing mechanism is suggested (see below).

4.2.7 Split-brain

The proposed architecture design is prone to split-blain scenarios, where two instances of a logistics system are running, that may lead to data inconsistency, data loss and data corruption. Therefore, a fencing mechanism that puts a node that is in an unknown state into a known state has to be implemented (for details see Subsection 3.1.4).

The proposed HA architecture assumes that the infrastructure layer provides independent mechanisms to power off a node with unknown state (e.g. smart rack power distribution units, lights-out management cards) using cluster fencing agents in order to prevent any uncoordinated actions within the cluster.

In a situation when all communication paths between two nodes are disrupted, a potential race condition may occurred when both nodes are trying to power off each other. To avoid this, fencing agents should be deployed with different delays before starting to fence the other node. Hence, the node which issues a fence command firstly will stay online. Once cluster software is assured that the node is powered off (i.e. brought to known state), a logistics system can be safely started on the surviving node. In terms of CAP theorem, the proposed approach trades off data availability for its consistency *when* partitioned (see details in Subsection 3.1.4).

5 EXPERIMENTAL SETUP

“Good problems and mushrooms of certain kinds have something in common; they grow in clusters.”
by George Pólya

This chapter presents an experimental setup used to validate the outlined HA architecture. Firstly, prototyping strategy for proposed HA architecture is describes. Preference for prototype implementation of cluster was given to free and open-source software running on commodity server hardware. Next, a task was set to test the working hypotheses on implemented prototype cluster by conducting experiments in the testbed. Further, captured qualitative and quantitative results from the testbed are presented. Finally, evaluation of the implemented prototype is done.

5.1 Prototyping Strategy

The reference HA cluster architecture was assessed for technological feasibility. This was done by mapping the concrete technology that should be used in HA cluster prototype to a functional layer within reference architecture (for details see Section 4.2).

5.1.1 Approach

The choice of technology was primary depend on technical requirements of KiSoft WCS which prototype implementation had to meet. Software implementations used for prototype were based on exclusively on free and open-source software to meet a non-functional requirement of cost-effectiveness of HA solution. Simplicity of technology was another crucial aspect during prototyping. Preference was given for the implementations that use the simplest possible approaches and algorithms to provide the needed level of availability.

Performed mapping of technologies to the reference architecture is shown in Figure 5-1:

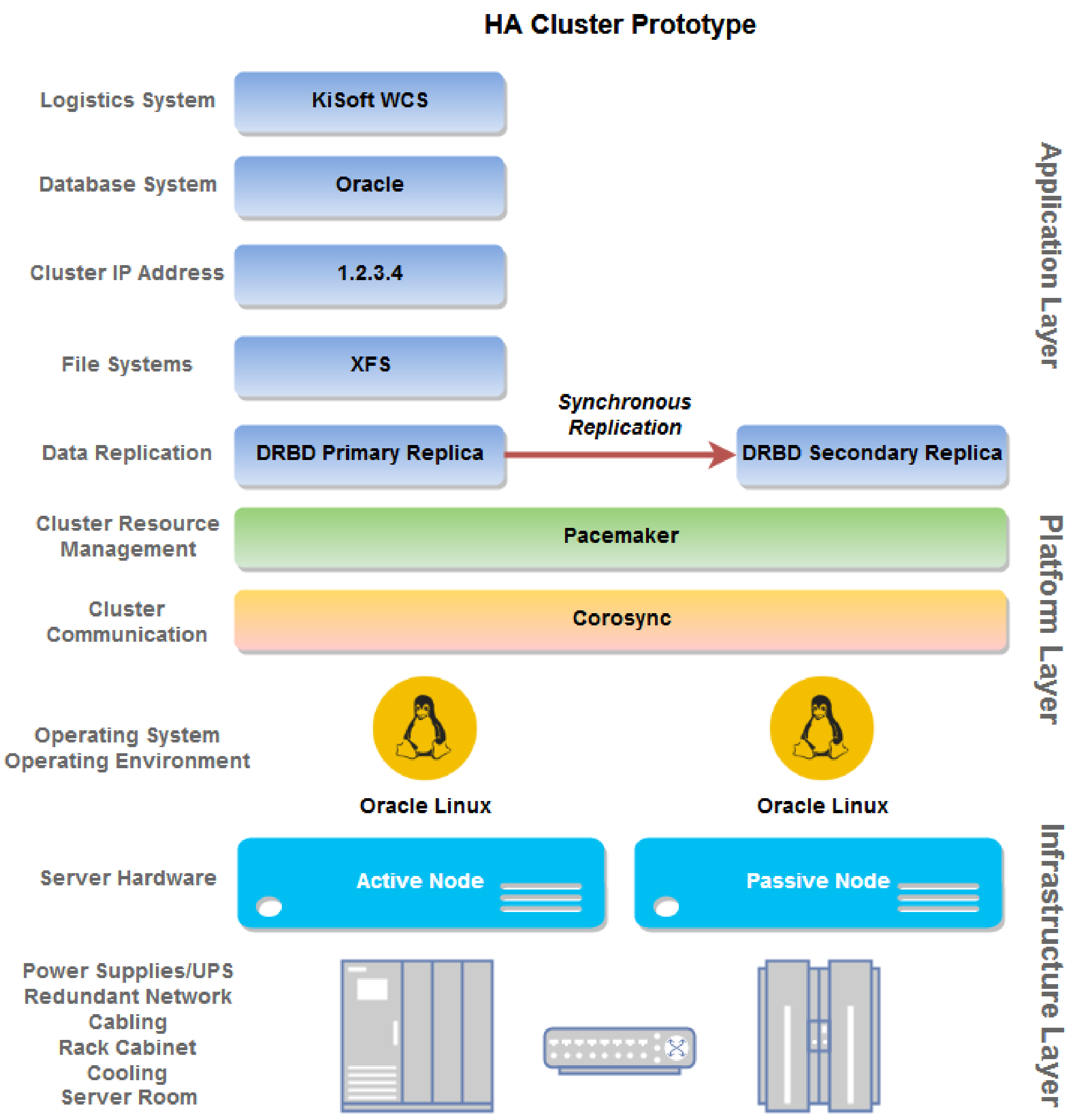


Figure 5-1: Prototype implementation of HA cluster

5.1.2 Hardware

In terms of technical means, the prototype cluster implementation was oriented on modern x86 general-purpose commodity server systems, which possess important hardware features for improving RAS (for details see Subsection 3.1.1).

Prototype implementation explicitly used the following RAS features of server hardware:

- watchdog timers
- IPMI implemented as integrated baseboard management controller (BMC) within lights-out management (LOM) products
- RAID technology

The hardware watchdog timer in the prototype was used to reboot cluster nodes if the operating system hanged to minimize potential outage. Hardware watchdogs, when integrated into server LOM devices, run independently of the operating system. Under normal operation, the watchdog driver in the operating system periodically resets the watchdog timer. If operating systems hang, the watchdog timer cannot be reset anymore, and thus after watchdog time expires LOM reboots the server (Dyke, Shaw, & Bach, 2011, p. 318).

Moreover, built-in LOM devices were used for *node-level fencing*. The LOM devices of both nodes were connected to the network of cluster nodes and were assigned IP addresses in order to communicate with fencing agents in the cluster software. In case of a communication failure between the nodes, fence agents which are configured with different delays on both nodes, would try to log in to each other's LOM and power off the partner node. The "fastest" node would stay online and can safely execute the workload of the logistics system (cf. (Muhamedagic, 2016)).

RAID was used to group the hard drives within two arrays, as shown in Table 5-1. The RAID 1 array was used to install the operating system and basic software needed for KiSoft WCS, whereas the RAID 1+0 array was used for KiSoft WCS itself and for the database (i.e. data files, control files and redo log files).

RAID Level	Number of Drives	Fault Tolerance	Usage
RAID 1 (mirror)	2	1-drive failure	Operating system and required software for KiSoft WCS
RAID 1+0 (mirror+stripe)	4	1-drive failure	KiSoft WCS and database data

Table 5-1: RAID arrays used in prototype

Moreover, one drive was added and configured as a global hot-spare to replace a failed disk in either of the two RAID arrays.

5.1.3 Networking

There was also an orientation to the Ethernet network architecture, namely Gigabit and 10 Gigabit Ethernet, which could provide fast and more reliable inter-node communication in the cluster at the physical level.

According to Sumimoto, et al. (1999) and Bakesa, Kimb, & Ramosb (2003), with introduction of Gigabit Ethernet in 1999, Ethernet-family technologies lost many of its initial "teething problems" related to loss of frames. Later, in 2002 and 2004, the most widely used standards of 10 Gigabit Ethernet were standardized. They increased bandwidths by using the new 64b/66b encoding (a rate of 10*64/64 gigabits per second) (IEEE, 2002).

Cluster interlink was implemented using 10 Gigabit Ethernet to provide the lowest latencies and bandwidth for synchronous replication. Uplink to warehouse subsystems network was built using Gigabit Ethernet.

Both, cluster interlink and uplink to warehouse subsystems network utilized dual network adapters that had two physical network connections. Moreover, they were aggregated in one logical network interface using bonding driver[32] of operating system.

Link aggregation was implemented using an *active-backup* mode of the boding driver. This mode implies that only one network interface controller (NIC) enslaved in the logical network interface is active at any time. If the active NIC fails, the other NIC takes over. (van Vugt, 2014, p. 28)

Active-backup mode for link aggregation was chosen as the simplest mode that provides fault tolerance. The other modes for link aggregation using a bonding driver, which were considered for prototype implementation, are shown in Table 5-2:

Mode	Use
balance-rr	This is the round-robin mode in which packets are transmitted in sequential order from the first network interface through the last.
active-backup	In this mode, only one slave is active, and the other slave takes over, if the active slave fails.
balance-xor	A mode that provides load balancing and fault tolerance and in which the same slave is used for each destination MAC address.
broadcast	This mode provides fault tolerance only and broadcasts packets on all slave interfaces.
802.3ad	This is the LACP mode that creates aggregation groups in which the same speed and duplex settings are used on all slaves. It requires additional configuration on the switch.
balance-tlb	In this mode, which is known as adaptive transmit load balancing, a packet goes out, according to load, on each network interface slave. Incoming traffic is received by a designated slave interface.
balance-alb	This works like *balance-tlb* but also load balances incoming packets.

Table 5-2: Network bonding modes (van Vugt, 2014, p. 28)

5.1.4 Operating System

Oracle Linux[33], a freely distributed distribution of GNU/Linux, was used as an operating system for prototype implementation. Since KiSoft WCS is tightly coupled to an Oracle database[34], the

[32] https://wiki.linuxfoundation.org/networking/bonding – Linux bonding driver

[33] https://www.oracle.com/linux/ – Oracle Linux

[34] https://www.oracle.com/database/ – Oracle Database

choice of Linux distribution for the prototype was based on Oracle preference on operating systems for their RDBMS (Oracle, n.d.). Moreover, Oracle Linux provides “High Availability Add-On”, which includes the required software for clustering.

Oracle Linux is shipped with Unbreakable Enterprise Kernel (UEK), upstream Linux kernel hardened with additional patches aimed to improve reliability, security, and performance. Specifically, Ksplice technology[35], provides live patching of operating system’s kernel without need to reboot, is supported by UEK (Casey, 2016).

5.1.5 Data Protection

According to Shivakumar (2014, p. 77), data is the most valuable part of any application, and thus data protection is a “cornerstone” in any highly available system.

To protect data for KiSoft WCS and database systems a synchronous master-slave data replication occurring between two nodes was used. The prototype cluster implemented block-layer replication, using *Distributed Replicated Block Device (DRBD)*[36].

DRBD operates at the generic block layer of an operating system and transparently replicates over the network all locally changed blocks (i.e. writes) from the active (in DRBD terms called as Primary) to the passive (Secondary) node, as shown in Figure 5-2. They are represented as block devices (e.g. `/dev/drbdX`) on both node. As only the active node may write to the replicated block device there is no need for a special clustered file system. Any file system that is designed to reside on the block device may be used over DRBD to support replication over the network. (Ellenberg, 2008)

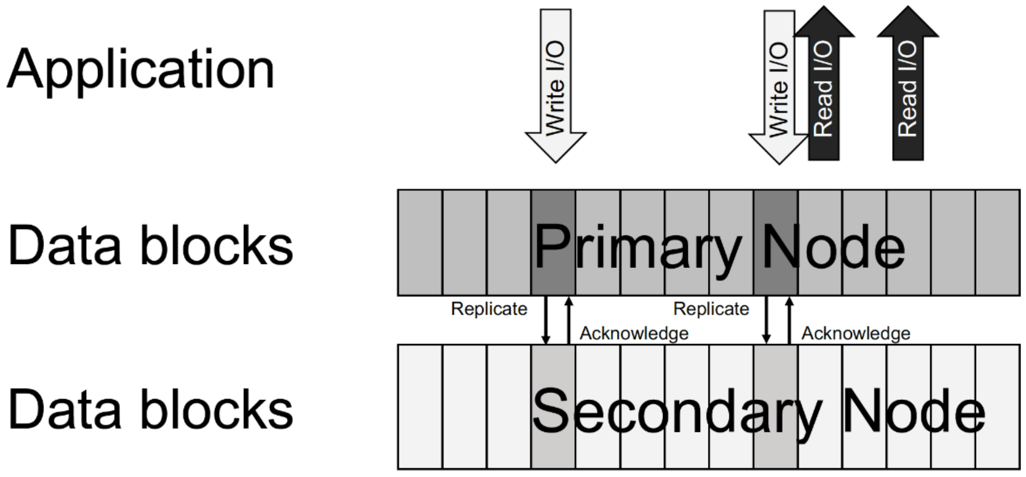


Figure 5-2: DRBD operation example, based on (Ellenberg, DRBD 9 - Linux Storage Replication, 2008)

DRBD implements three replication modes, described in Table 5-3:

[35] http://ksplice.oracle.com – Oracle Ksplice

[36] https://www.drbd.org/ – DRBD brings you High Availability and Disaster Recovery

Protocol	Type of replication	Description
A	*Asynchronous replication*	Local write operations on the primary node are considered completed as soon as the local disk write has finished, and the replication packet has been placed in the local TCP send buffer. In the event of forced fail-over, data loss may occur. The data on the standby node is consistent after fail-over, however, the most recent updates performed prior to the crash could be lost.
B	*Semi-synchronous replication*	Local write operations on the primary node are considered completed as soon as the local disk write has occurred, and the replication packet has reached the peer node. Normally, no writes are lost in case of forced fail-over. However, in the event of simultaneous power failure on both nodes and concurrent, irreversible destruction of the primary's data store, the most recent writes completed on the primary may be lost.
C	*Synchronous replication*	Local write operations on the primary node are considered completed only after both the local and the remote disk write have been confirmed. As a result, loss of a single node is guaranteed not to lead to any data loss. Data loss is, of course, inevitable even with this replication protocol if both nodes (or their storage subsystems) are irreversibly destroyed at the same time.

Table 5-3: Replication modes for DRBD-backed SDS (LINBIT, 2016)

To pursue the RTO objective of maximal data protection in the prototype cluster *Protocol C* for truly synchronous replication was investigated.

An abstraction for block devices in the Linux kernel made possible to create files systems on top of DRBD device. DRBD devices, in its turn, were created on top of logical volumes, combined within volume group. Volume groups were resided on top of block devices that represented RAID arrays.

Such an approach to use DRBD as backed block device for the specific file system allowed to replicate only the data which were related to KiSoft WCS and Oracle Database.

Table 5-4 presents abstraction layers used to build a resilient replicable software-defined storage for prototype cluster:

Block Device	**Volume Group**	**Logical volume**	**DRBD device**	**Mount point**	**Restrictions**
/dev/sda (RAID 1)	rootvg	root	not used	/	node-specific data, mounted on both nodes
		swap		not needed	
		home		/home	
		opt		/opt	
		var		/var	
		tmp		/tmp	
/dev/sdb (RAID 1+0)	extvg	kisoft-wcs	/dev/drbd0	/kisoft/wcs	mounted on active node
		kisoft-dbdata	/dev/drbd1	/kisoft/dbdata	

Table 5-4: Levels of abstractions for resilient storage, which were used in prototype

To speed up the process of file system integrity check and recovery in case of failover caused by a node crash, XFS[37], a journaling file system, was chosen for prototyping. In addition, XFS is a POSIX-compliant file system[38], which implies that no application change is needed.

5.1.6 Cluster Setup

There are two main open source projects: Corosync[39] and Pacemaker[40], consolidated under the Cluster Labs[41] umbrella, with a goal to develop a solution for creating high availability systems based on clusters running GNU/Linux operating system. They are the latest implementations of HA clustering software under GNU/Linux that follows the specifications of Open Cluster Framework (OCF)[42] (van Vugt, 2014, p. 4). They form, so called, Linux-HA cluster stack, as seen in Figure 5-3, which besides Corosync and Pacemaker includes application adaptor scripts (*resource agents*), fencing adaptors (*fencing agents*) and command line-tools to configure all of them (*pcs*) (Schönig, 2015, p. 132).

[37] http://xfs.org/

[38] http://www.sgi.com/products/storage/software/xfs.html

[39] http://corosync.github.io/corosync/ – The Corosync Cluster Engine

[40] http://clusterlabs.org/pacemaker.html – Pacemaker: A scalable High-Availability cluster resource manager

[41] http://clusterlabs.org/ – The Home of Linux Clustering

[42] http://www.opencf.org/ – Open Cluster Framework

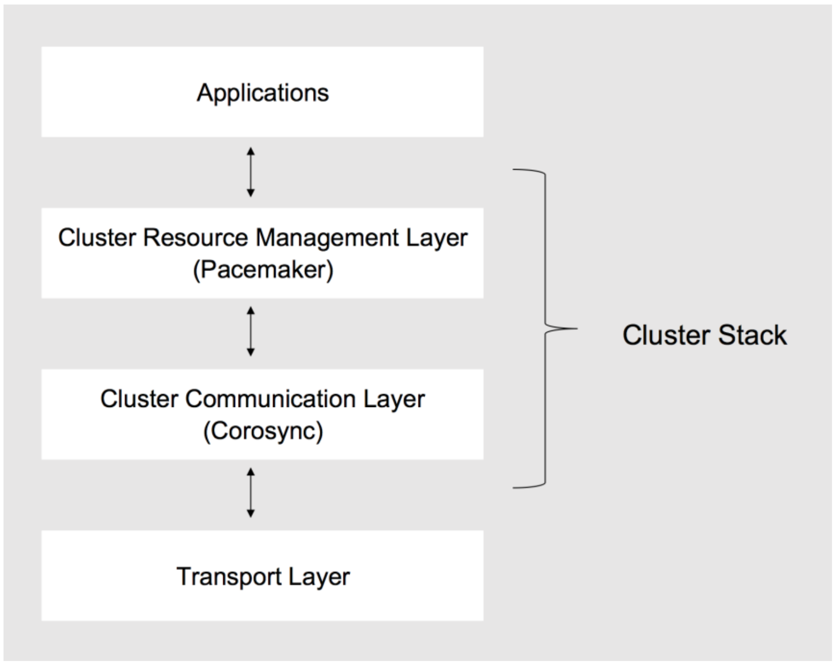

Applications
Cluster Resource Management Layer (Pacemaker)
Cluster Communication Layer (Corosync)
Transport Layer
Cluster Stack

Figure 5-3: Linux HA cluster stack, based on (Resman, 2015, p. 11)

Corosync provides group communication between cluster nodes ensuring the virtual synchrony property (for details see Subsection 3.1.4) across all cluster nodes (i.e. consistent ordered delivery of messages). In this regard Corosync implements Totem Single-ring Ordering and Membership (TOTEM) protocol for cluster messaging and membership services (cf. (Ciarfella, Moser, Melliar-Smith, & Agarwal, 1994). Cluster nodes form a *ring* where a “heartbeat” token is consistently passed between the nodes. Only one node in a cluster that possesses this token can broadcast messages to other cluster nodes. Each message has a sequence number to guarantee ordered message delivery. If a cluster node does not pass the token around the ring after token timeout for several repetitions it will declared as failed and will be removed from the cluster. The remaining cluster nodes will try to rebuild the cluster configuration. If quorum can be achieved the new cluster configuration will be activated. (Resman, 2015, pp. 12-14, 20)

For redundant heartbeat network Corosync supports the use of multiple networks. Nevertheless, in the process of cluster prototyping it was discovered that latest stable (and shipped with Oracle Linux 7.3) version of Corosync 2.4 was limited to two logical token-passing rings. This restriction was not covered in supplied documentation, but, it fact, it was not possible to configure Corosync to use more than two logical network interfaces due to fatal errors in the process of daemon’s start (as such limit was hardcoded in source code[43]). Therefore, it was possible to implement only 1+1 redundancy for cluster membership and messaging in the prototype cluster.

Corosync options, “`two_node`” and “`wait_for_all`”, were used to deal with the quorum dilemma in two-node cluster (for details see Subsection 3.1.4). The first option deactivates quorum, thus allowing “one-node cluster” to operate if the second node fails (i.e. during failover) (Beekhof, 2015, p. 30). The second option prevents the start of cluster resources when both nodes were down

[43] https://github.com/corosync/corosync/blob/0462b5e609c95c0187e157b796c605988f16b784/exec/totemconfig.c#L1052

and only one node booted up (i.e. power fencing loop) to avoid possible data loss or data corruption caused by a "split brain" situation (cf. (Caulfield, 2016, pp. 2-3)).

In its turn, *Pacemaker* is a cluster resource manager that manages and monitors services and applications, called *cluster resources*, in the cluster. Pacemaker relies on information provided by Corosync, the bottom cluster communication layer, about the cluster nodes and their status. Based on this information cluster resource actions are taken (e.g. to restart a crashed cluster resource or to migrate cluster resources to operational cluster node if a node failure occurs). (Resman, 2015, pp. 10, 16)

Cluster resources are implemented as *resource agents*. Several resource agent standards are supported, including OCF and LSB[44] scripts, along with systemd[45] unit files. Additionally, cluster resources can be combined into resource groups and include constraints for location, order or collocation to improve manageability. (Beekhof, 2015, pp. 27-28; Resman, 2015, pp. 110-113)

Implemented prototype cluster Pacemaker was configured with minimum needed cluster resources to run KiSoft WCS. Cluster resources were combined into resource groups, as shown in Table 5-5, which include:

- A master/slave resource set with a resource group named "`g_drbd`" containing two DRBD resources for synchronous block-based data replication of logistics and database systems. DRBD resource is promoted as master only on the active node.
- Resource group "`g_cluster_res`" with prerequisites for KiSoft WCS, including resource agents to mount file systems on top of DRBD devices, Oracle Database and cluster IP address.
- Resource group "`g_wcs`" with KiSoft WCS that is started as the last resource in a cluster.

Order	Resource Group	Master/Slave Set	Resource Agents
1	g_drbd	Yes	DRBD resource for KiSoft WCS file system
2			DRBD resource for Oracle Database file system
3	g_cluster_res	No	File system for KiSoft WCS
4			File system for Oracle Database
5			Oracle Database
6			Cluster IP address
7	g_wcs	No	KiSoft WCS

Table 5-5: Cluster resources in prototype cluster

[44] http://refspecs.linuxfoundation.org/lsb.shtml

[45] https://freedesktop.org/wiki/Software/systemd/ – systemd - System and Service Manager

Finally, a colocation and order constraints were added to ensure that all cluster resources in prototype cluster were started in specific order and running together on the same cluster node.

5.2 Testbed

This section of the thesis introduces the test bedding approach for the prototype, describes the testbed setup, and defines the experiments that were performed.

5.2.1 Approach

A two-node cluster was set up as the testbed in accordance with the prototyping strategy. Experiments to test the working hypothesis were outlined, and the resulting experiments yielded both qualitative and quantitate information. Qualitative data was derived from observations of prototype operation under simulated conditions. In its turn, measurements of failover performance and data protection were executed to provide quantitative data in order to determine if target HA objectives (i.e. RTO and RPO in Table 4-1) were achieved.

5.2.2 Testbed Setup

The prototype cluster was set up as testbed using equipment and server rooms provided by KNAPP that had access their real scale warehouse testbed.

Firstly, RAID controllers on both servers were configured with the same RAID layout and each LOM was assigned an IP address along with a special user account for the power fencing agent.

Next, Oracle Linux operating system was provisioned using the Anaconda Kickstart[46] installation method (for details see APPENDIX A - Anaconda Kickstart File). Hostnames were set to `A001-TEST-Graz-SRV1` (hostname alias “`node-1`”) and `A001-TEST-Graz-SRV2` (hostname alias “`node-2`”) accordingly. Each network interface was assigned with unique IP addresses. Cluster interlink was deployed as dual point-to-point connections. The other network interface was redundantly connected to the warehouse test network over network switches in server racks. The network topology used for testbed is shown in Figure 5-4:

[46] http://pykickstart.readthedocs.io/en/latest/kickstart-docs.html – http://pykickstart.readthedocs.io/en/latest/

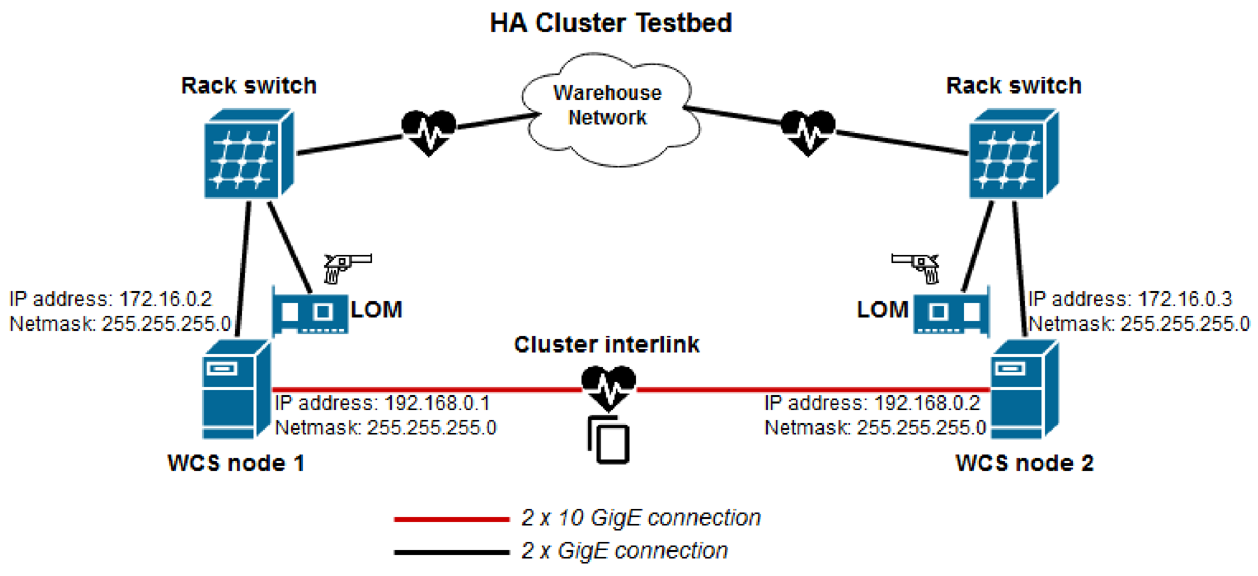


Figure 5-4: Network topology in testbed

System clocks on the nodes were synchronized using NTP protocol to preserve correct time stamps during experiments and SSH key-based authentication was configured to perform semi-automatic testing using Shell-scripts.

Further, Oracle Database was installed and a database instance for KiSoft WCS was created. The latest KiSoft WCS version was installed and configured.

Cluster and resilient software-defined storage configurations were deployed automatically with Shell-scripts (for details see APPENDIX B - HA Cluster Setup and APPENDIX C - Resilient Storage Setup).

Corosync was configured to use two rings for cluster “heartbeating” process, as shown in Listing 5-1, reflecting the network topology in Figure 5-4:

```
$ corosync-cfgtool -s
Printing ring status.
Local node ID 1
RING ID 0
        id      = 172.16.0.2
        status  = ring 0 active with no faults
RING ID 1
        id      = 192.168.0.1
        status  = ring 1 active with no faults
```

Listing 5-1: Status of Corosync “heartbeat” rings

The output of the `pcs status`[47] command, as seen in Listing 5-2, captures the prerequisite status of cluster before proceeding to perform any experiment:

[47] http://clusterlabs.org/doc/en-US/Pacemaker/1.1/html/Clusters_from_Scratch/_explore_pcs.html

```
[root@A001-TEST-Graz-SRV1|>~]$ pcs status
Last updated: Wed Jun 21 14:53:10 2017       Last change: Wed Jun 21
10:56:54 2017 by root via cibadmin on A001-TEST-Graz-SRV1
Stack: corosync
Current DC: A001-TEST-Graz-SRV1 (version 1.1.15-11.el7_3.2-e174ec8) - par-
tition with quorum

Online: [ A001-TEST-Graz-SRV1 A001-TEST-Graz-SRV2 ]

 Master/Slave Set: ms_drbd [g_drbd]
     Masters: [ A001-TEST-Graz-SRV1 ]
     Slaves: [ A001-TEST-Graz-SRV2 ]
 Resource Group: g_cluster_res
     p_fs_kisoft_dbdata     (ocf::heartbeat:Filesystem):  Started A001-
TEST-Graz-SRV1
     p_fs_kisoft_wcs  (ocf::heartbeat:Filesystem):  Started A001-TEST-
Graz-SRV1
     p_oracledb (systemd:oracledb):     Started A001-TEST-Graz-SRV1
     p_vip_knapp      (ocf::heartbeat:IPaddr2):     Started A001-TEST-
Graz-SRV1
 Resource Group: g_wcs
     p_kisoft_wcs     (systemd:wcs):    Started A001-TEST-Graz-SRV1

PCSD Status:
  A001-TEST-Graz-SRV1: Online
  A001-TEST-Graz-SRV2: Online

Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled
  pcsd: active/enabled
```

Listing 5-2: Status of HA cluster before conducting any experiment

The output of the `drbd-overview`[48] command, reference status of configured DRBD resources is shown in Listing 5-3:

[48] https://docs.linbit.com/doc/users-guide-84/ch-admin/#s-drbd-overview

```
$ drbd-overview
 0:kisoft-dbdata/0  Connected Primary/Secondary UpToDate/UpToDate
/kisoft/dbdata xfs 60G 15G  46G 25%
 1:kisoft-wcs/0     Connected Primary/Secondary UpToDate/UpToDate
/kisoft/wcs  xfs 40G 6.4G 34G 16%
```

Listing 5-3: Status of DRBD devices before conducting any experiment

A summary of components that were used in testbed is listed in Table 5-6:

Testbed Component	**Description**
Physical environment	Two 19" rack cabinets located in separate server rooms. Distance between server rooms is about 50 meters
Hardware	Two HPE ProLiant ML350 Gen9 servers[49] with integrated iLO, lights-out management card (LOM)[50] that was connected to a warehouse network switch
Network connectivity	Each server node had: ▪ bonded (2 x 10 GigE) point-to-point connection between nodes using multi-mode optical fiber dedicated to synchronous data replication and cluster "heartbeats" ▪ bonded (2 x 1 GigE) connection to a test warehouse network switch used by KiSoft WCS to communicate with, by Corosync for "cluster "heartbeats" and by Pacemaker fence agent to power off the node with unknown state using its LOM ▪ a network latency for all network connections less than 500 µs
Directly-attached storage	2 RAID arrays: ▪ RAID 1 (mirror) made of 2 disk drives and used for operating system partition ▪ RAID 1+0 (stripe of mirrors) made of 4 disk drives and used for the partitions of the logistics systems and its database 1 global "hotspare" disk drive
Operating system	Oracle Linux 7.3 x86-64 with Unbreakable Enterprise Kernel 4.1.12
Software-defined replicated data storage	DRBD 8.4.8

[49] https://www.hpe.com/h20195/v2/getpdf.aspx/c04375628.pdf – HPE ProLiant ML350 Generation9 (Gen9) QuickSpecs

[50] https://www.hpe.com/emea_europe/en/servers/integrated-lights-out-ilo.html – HPE Integrated Lights Out (iLO) Server Management

Partition layout	▪ `/`, `/home`, `/opt`, `/var`, `/tmp` - mount points created during operating system installation ▪ `/kisoft/wcs` – mount point for files of the logistics system, formatted as XFS file system and sycnhroniously replicated by DRBD ▪ `/kisoft/dbdata` – mount point for database, formatted as XFS file system and synchronously replicated by DRBD
HA clustering software	Corosync 2.4.0 Pacemaker 1.1.15
Database system	Oracle Database 12c Release 2
Logistics system	KiSoft WCS 8.0

Table 5-6: Testbed components

5.2.3 Performance Test

To investigate the overall performance capabilities of the prototype cluster the following test scenarios with expected results, as shown in Table 5-7, were examined:

Test Scenario	Expected Result
Shutdown of passive node	Warehouse operation is to not be affected.
Reboot of passive node	Warehouse operation is to not be affected.
Shutdown of active node	Automatic switchover is triggered. Warehouse operation is affected until automatic switchover is finished.
Reboot of active node	Automatic switchover is triggered. Warehouse operation is affected until automatic switchover is finished.
Manual switchover	Warehouse operation is affected until manual switchover is finished.

Table 5-7: Overall performance tests for prototype cluster with expected results

Shutdown and reboot actions were initiated in graceful way using `systemctl poweroff` and `systemctl reboot` commands accordingly. Manual switchover was performed using special test script, provided in Listing 5-4, that encapsulated several commands to migrate cluster resources from one node to other.

```
#!/bin/bash

# Cluster resources
HA_RESOURCES=( g_cluster_res g_wcs )

# Helper functions
crm_check_up() {
  pcs status >/dev/null 2>&1 || return 1
}

crm_who_am_i() {
  local_node=$(crm_node -n 2> /dev/null)
}

crm_node_count() {
  node_count_all=$(crm_node_list | wc -w)
}

crm_node_online() {
  local node_online
  node_online=$(crm_node --partition 2> /dev/null)
  echo -n "$node_online"
}

crm_constraints_clean() {
  # Clear temporary location constraints if any but fencing and user-de-
fined location constraints
  OIFS=$IFS
  IFS=$'\n'
  loc_constraints=$(cibadmin -Q 2> /dev/null | grep -e "<rsc_location " |
grep -v stonith | grep -v location- | grep -v l_ | sed
's/[[:space:]]\{2,\}//g')

  for i in $loc_constraints; do
    cibadmin --delete --xml-text "$i" 2> /dev/null
  done
  IFS=$OIFS
}

crm_resource_cleanup() {
  local node
  local failed_res
  local fail_value

  node=$(crm_node -l 2> /dev/null | cut -d' ' -f 2)
  failed_res=$(crm_mon -f1 2> /dev/null | grep fail-count | awk '{ print
$1}' | awk '{print (substr($1,0,length($1)-1))}')
```

```
  for res in $failed_res; do
    for i in $node; do
      fail_value=$(crm resource failcount $res show $i 2> /dev/null | cut
-d' ' -f4 | cut -d= -f2)
      if [[ $fail_value == INFINITY ]]; then
        crm resource cleanup $res $i > /dev/null 2>&1
      else
        if [[ $fail_value != 0 ]]; then
          crm resource failcount $res delete $i > /dev/null 2>&1
        fi
      fi
    done
  done
}

crm_resource_unmove() {
  for resource in "${HA_RESOURCES[@]}"; do
    crm_resource -r $resource -U > /dev/null 2>&1
  done
  mapfile -t resources < <(crm_resource -L 2> /dev/null | awk '!/Resource
Group:/ {print $1}' | sed -e '/Master\/Slave/,$d')
  for resource in "${resources[@]}"; do
    crm_resource --resource "$resource" --un-move > /dev/null 2>&1
  done
}

timestamp() {
  timestamp=$(date '+%d-%m-%Y %H:%M:%S')
}

if crm_check_up; then
  crm_node_count
  if [[ "$node_count_all" -ne 2 ]]; then
    echo "ERROR: Unsupported cluster setup"
    exit 1
  fi
  crm_who_am_i
  # Clearing temporary location constraints
  crm_constraints_clean
  crm_resource_cleanup
  other_node=$(crm_node_list | grep -v $local_node)
  # Execute resource move
  for resource in "${HA_RESOURCES[@]}"; do
    echo "Moving $resource"
    crm_resource -f --wait -r "$resource" -M -Q -N "$other_node" &
procs="$! $procs" > /dev/null 2>&1
```

```
      wait $procs
      if [ $? -ne 0 ]; then
        rc=$?
        break
      fi
      rc=0
    done
    # Waiting for stable status of cluster or timeout
    dc=$(crm_list_dc)
    timeout=300
    cnt=0
    until crmadmin -S "$dc" -t 10000 | grep -qs S_IDLE 2>/dev/null || [ $cnt
  -gt $timeout ]; do
      sleep 1
      cnt=$((cnt+1))
    done
    if [[ "$rc" -eq 0 ]] && [[ $(crm_mon -1r | grep -cE 'Stopped|Failed') -
  eq 0 && $(crm_list_master) != "$local_node" ]]; then
      # Clearing temporary location constraints & resources after move...
      crm_constraints_clean
      crm_resource_cleanup
      # Un-migrate resources after successful migration
      crm_resource_unmove
      timestamp
      echo -e "RESULT:\n"
      pcs status
      echo -e "\nMigration was successful at $timestamp"
      echo "=================================================="
      exit 0
    else
      crm_constraints_clean
      crm_resource_cleanup
      timestamp
      echo -e "RESULT:\n"
      pcs status
      echo -e "\nMigration failed at $timestamp"
      echo "=================================================="
      exit 1
    fi
  fi
```

Listing 5-4: Switchover test script

5.2.4 Failure Simulation

Based on failure statistics from Subsection 4.1.2, the following failures, listed in Table 5-8, were simulated:

Failure	Approach
Crash of logistics system	Stop KiSoft WCS in a graceless way (i.e. execute `kill -SIGKILL $(pidof king`[51]`)` on active node).
Hard drive failure	Remove any non-hot-spare drive from active node while it is running.
RAID array failure	Remove all hard drives from active node.
Network failure	Remove any network cable from active node.
Split-brain	Disconnect all network cables from to prevent internode communication.

Table 5-8: Simulated failures in testbed

5.2.5 Failover Performance

Node crash and failover performance were investigated additionally. Failover performance was measured a during the time between a crash of active node and start of KiSoft WCS on the passive node.

System crash of the active node was simulated using random time intervals between 1 and 300 seconds. Character “c” was written to `sysrq-trigger` file on `procfs`, virtual file system that is used for low-level communication between kernel space and user space without using special system calls (cf. (Kerrisk, 2010, pp. 221-228)).

Next, the time between system crash and readiness of KiSoft WCS to accept connections on TCP port 9801 used for communication between other subsystems in the warehouse was measured. Netcat[52] networking utility was used to check readiness of socket to accept connection requests.

In the cases where the failover process was not triggered automatically or took longer than five minutes, it was declared as unsuccessful. Such quantitative data from this experiment was also used to measure *Automatic Failover Success Rate* for the cluster setup.

Shell-scripts from Listing 5-5 was used to conduct the experiment one hundred times:

[51] king – is the master process of KiSoft WCS

[52] http://nc110.sourceforge.net/ – Netcat is a simple Unix utility which reads and writes data across network connections, using TCP or UDP protocol.

```
#!/bin/bash

#
# Failover Test
#

echo "Test started at $(date +%Y/%m/%d-%H:%M:%S)"

# Determine the hostnames of the nodes
CURHN=$(hostname -s)

if [[ $CURHN =~ ^[A-Za-z0-9-]{2,}[1]$ ]]; then
  ME="node-1"
  PEER="node-2"
else
  ME="node-2"
  PEER="node-1"
fi

# TCP port KiSoft WCS listening on
WCS_PORT="9801"

# Helper functions for sanity checks
is_cluster() {
  pcs status > /dev/null 2>&1 || return 1
}

is_master() {
  local mnt="$1"

  if grep -q "drbd.*$mnt " /proc/mounts; then
    return 0
  else
    return 1
  fi
}

is_wcs_running() {
  pcs resource status p_wcs | grep -q "Running" || return 1
}

if is_cluser || ! is_mounted "/kisoft/wcs" && is_wcs_running; then
  time=$(((RANDOM%299)+1))
  sleep $time
  # Simulate a system crash by rebooting actvie node in a graceless way
  ssh root@$PEER "echo 1 > /proc/sys/kernel/sysrq; echo c > /proc/sysrq-
trigger" &
```

```
    FAILOVER_START=$(date +%s)
    # Measuring failover time
    # Give up after 5 minutes and report failed failover
    timeout=300
    cnt=0
    until nc -v $ME $WCS_PORT 2>&1 | grep -q "Connected to" || [ $cnt -ge
  $timeout ]; do
      sleep 1
      cnt=$((cnt+1))
    done
    if nc -v $ME $WCS_PORT 2>&1 | grep -q "Connected to"; then
      FAILOVER_END=$(date +%s)
      FAILOVER_TIME=$(( FAILOVER_END - FAILOVER_START ))
      echo "Failover time: $FAILOVER_TIME seconds" >> failover_time.txt
    else
      echo "Failover failed" >> failover_fail.txt
    fi
  else
    echo "Run this test on passive node while WCS is running on primary
  node"
  fi

  echo "Test finished at $(date +%Y/%m/%d-%H:%M:%S)"
```

Listing 5-5: Failover test script

5.2.6 Data Protection

To test data protection within the cluster the following approach was used. On the active node randomly generated data from Linux kernel's pseudorandom number generator interface `/dev/urandom` (cf. (Love, 2013, p. 281)) was redirected to file `testdata` located in `/kisoft/wcs` mount point using `dd`[53] command line utility with `O_DIRECT` and `O_SYNC` flags to bypass the operating system write and write cache and write data synchronously to disk (cf. (Kerrisk, 2010, pp. 241-242, 246-248)).

Next, in a random time interval between 1 and 300 seconds a system crash of active node was simulated by written "c" to `/proc/sysrq-trigger`. Immediately, cluster interlink was disabled to prevent data replication and the cluster was put in maintenance mode. When failover was finished, existence of `/kisoft/wcs/testdata` file on the node that took over was verified. If the file

[53] https://www.gnu.org/software/coreutils/manual/html_node/dd-invocation.html – dd: Convert and copy a file

existed, SHA-256[54] hashing algorithm was used to get a checksum of the file. Otherwise, a data loss was detected and recorded.

Further, DRBD resource on crashed node that held data for `/kisoft/wcs` mount point was examined the same way to get SHA-256 checksum of the file.

Finally, obtained checksums for `/kisoft/wcs/testdata` file from both nodes were compared. If two checksums were identical no inconsistency of data between the two nodes was detected.

This experiment was performed one hundred times using a Shell-script, as shown in Listing 5-6:

[54] Secure Hash Algorithm 2 with 256-bit hash value

```
#!/bin/bash

# Data availability and consistency check

echo "Test started at $(date +%Y/%m/%d-%H:%M:%S)"

INTERLINK="bond1"

# Determine the hostnames of the nodes
CURHN=$(hostname -s)

if [[ $CURHN =~ ^[A-Za-z0-9-]{2,}[1]$ ]]; then
  PEER="node-2"
  PEER_WAREHOUSE="A001-TEST-Graz-SRV2"
else
  PEER="node-1"
  PEER_WAREHOUSE="A001-TEST-Graz-SRV1"
fi

# Helper functions for sanity checks
is_cluster() {
  pcs status > /dev/null 2>&1 || return 1
}

is_master() {
  local mnt="$1"

  if grep -q "drbd.*$mnt " /proc/mounts; then
    return 0
  else
    return 1
  fi
}

is_wcs_running() {
  pcs resource status p_wcs | grep -q "Running" || return 1
}

if is_cluser || ! is_mounted /kisoft/wcs && is_wcs_running; then
  # Begin to write some random data to the file on active node
  ssh $PEER "nohup dd if=/dev/urandom of=/kisoft/wcs/testdata conv=fdata-
sync oflag=direct,sync &"
  # Simulate a node crash during the write operation
  time=$(((RANDOM%299)+1))
  sleep $time
  ssh root@$PEER "echo 1 > /proc/sys/kernel/sysrq; echo c > /proc/sysrq-
trigger" &
```

```
  # Disable cluster interlink
  ifconfig $INTERLINK 0.0.0.0
  ifconfig $INTERLINK down

  # Wait until the mount point with the testfile failed over
  # Give up after 5 minutes
  timeout=300
  cnt=0
  until is_mounted "$TEST_MOUNT" || [ $cnt -ge $timeout ]; do
    sleep 1
    cnt=$((cnt+1))
  done
  if is_cluser || is_mounted "$TEST_MOUNT"; then
    checksum=$(sha256sum /kisoft/wcs/testdata)
    # Get checksum of the test file there
    echo "===="
    echo "$checksum"
    sha256sum /kisoft/wcs/testdata >> checksums.txt || echo "Data loss!!!"
>> data_loss.txt && exit 1
    if is_wcs_running; then
      # Stop KiSoft WCS and unmount /kisoft/wcs
      pcs resource stop p_kisoft_wcs > /dev/null 2>&1
      pcs resource stop p_fs_kisoft > /dev/null 2>&1
    fi
    # Put the cluster in maintenance mode in order
    # to mount replicated DRBD resource on the other node
    pcs property set maintenance-mode=true
    # Set DRBD resource to secondary to be able to
    # mount it on the other node
    drbdadm secondary kisoft-wcs
    # Log in to the other node over warehouse network (as interlink is
down)
    # Set the DRBD resource primary
    # Mount it
    # Get sha256 checksum
    ssh root@$PEER_WAREHOUSE bash -l -c "'
drbdadm primary kisoft-wcs
mount /kisoft/wcs
sha256sum /kisoft/wcs/testdata
'" >> checksums.txt
    echo "===="
    else
      echo "Failover is failed"
  fi
else
```

```
  echo "Run this test on passive node while WCS is running on primary
node"
fi
echo "Test finished at $(date +%Y/%m/%d-%H:%M:%S)"
```

Listing 5-6: Data availability and consistency test script

5.3 Results

This section presents qualitative and quantitative results of conducted experiments over the prototype testbed setup.

5.3.1 Overall Performance

The observations during the performance tests of the prototype cluster are captured in Table 5-9:

Test Scenario	Result of Observation
Shutdown of passive node	▪ No change of cluster resources occurred. ▪ Data replication was stopped.
Reboot of passive node	▪ No change of cluster resources occurred. ▪ Data replication was stopped. ▪ Once the passive node was up again, data replication resumed. The status of cluster resources remained the same.
Shutdown of active node	▪ Cluster resources were stopped on active node. ▪ Data replication was stopped. ▪ Cluster resources migrated to passive node and started there.
Reboot of active node	▪ Cluster resources were stopped on active node. ▪ Data replication was stopped. ▪ Cluster resources migrated to passive node and started there. ▪ Once the other node was up again, data replication resumed. The status of the active node remained the same (i.e. failback did not occurred).
Manual switchover	▪ Cluster resources were stopped on active node.

	▪ Data replication was stopped. ▪ Cluster resources migrated to passive node and started there. ▪ Once cluster resources were migrated to other node, data replication resumed.

Table 5-9: Results of performance tests

5.3.2 Fault Resilience

Table 5-8 presents observations during failure simulations from Subsection 0:

Failure	Result of Observation
Crash of logistics system	▪ Warehouse operation is interrupted. ▪ Cluster resource manager detected a crash of the KiSoft WCS when its status polling timer expired. ▪ Cluster resource manager started it again on the same node as before. ▪ TCP/IP stack of warehouse subsystems detected and handled connection failure. ▪ Warehouse operation were resumed. ▪ No manual intervention was needed.
Hard drive failure	▪ RAID controller detected a failed disk. ▪ Global hot-spare disk was automatically added in degraded RAID array and its rebuild was started. ▪ Failure was transparent for warehouse operation. ▪ No manual intervention was needed. ▪ No data loss occurred.
Local storage failure	▪ DRBD detected a “black-out” of locally attached storage. ▪ DRBD set the status of DRBD resource on active node as “`Diskless`”. ▪ DRBD transparently redirected all I/O activities to the passive node. ▪ The failure was transparent for warehouse operation.

	▪ No manual intervention was needed at first place. Eventually, to repair diskless node a switchover to the node with a "healthy" storage was conducted. ▪ No data loss occurred.
Network failure	▪ Bonding driver immediately detected a link failure and reconfigured active slave NIC ▪ The failure was transparent for warehouse operation. ▪ No manual intervention was needed. ▪ No network outage or packet loss occurred.
Split-brain	▪ Warehouse operation was interrupted. ▪ One node ("victim") was power off. ▪ The other node ("survivor") stayed online. However, all cluster resources were stopped. ▪ Network cable to warehouse network was attached. ▪ "Survivor" node started cluster resource along with KiSoft WCS. ▪ TCP/IP stack of warehouse subsystems detected and handled connection failure. ▪ Warehouse operation was resumed. ▪ No manual intervention was needed. ▪ "Victim" node had to be brought in operation manually. ▪ No data loss or data inconsistency occurred.

Table 5-10: Results of observations after conducting fault resilience tests

5.3.3 Average Failover Time

The formula in Equation 5-1 was used to calculate average failover time:

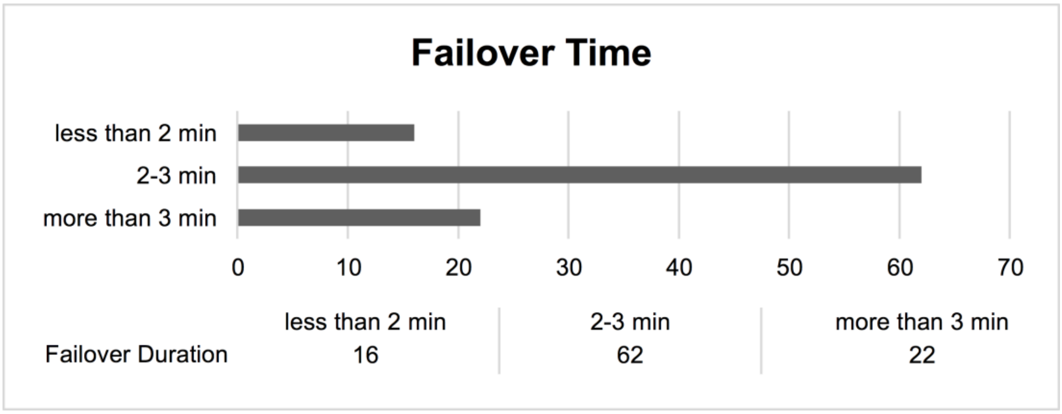
$$Avarage\ failover\ time = \frac{\sum_{i=1}^{N} x_i}{N}$$

Equation 5-1: Average failover time

where N is the number of conducted failover tests and x_i represents the measured time needed for each failover.

Average failover time for KiSoft WCS was based on the average time of one hundred failovers was recorded, and equaled 147.13 seconds.

Figure 5-5 plots time distribution of one hundred failover times:



	less than 2 min	2-3 min	more than 3 min
Failover Duration	16	62	22

Figure 5-5: Distribution of failover times triggered by hundred active node crashes

The shortest failover took one hundred seconds, whereas the longest failover that was recorded took 198 seconds. Most of failovers were no longer than three minutes.

5.3.4 Automatic Failover Success Rate

During the failover performance experiment one hundred failovers were triggered automatically by cluster software after a system crash of the active node was detected. KiSoft WCS was started on the healthy mode and was ready to accept remote connections on TCP port 9801 without any manual intervention. Each triggered failover was finished successfully. Hence, *automatic failover success rate* was 100%.

5.3.5 Data Availability and Its Consistency

Results of data protection experiment, which is visually represented in Figure 5-6, did not detect any data loss during write activities on the active node while randomly crashing the system.

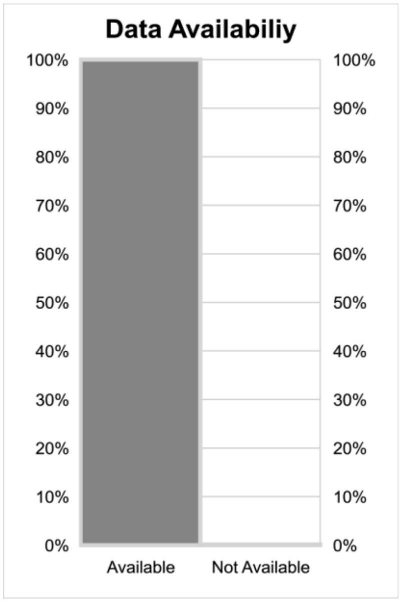


Figure 5-6: Percentage of available/unavailable data after a system crash

In its turn, results of comparison of one hundred SHA-256 checksums of the tested files between split nodes (i.e. replication was interrupted), as seen in the Figure 5-7, did not reveal any inconsistencies in data after the system crash of the active node, which was performed one hundred times.

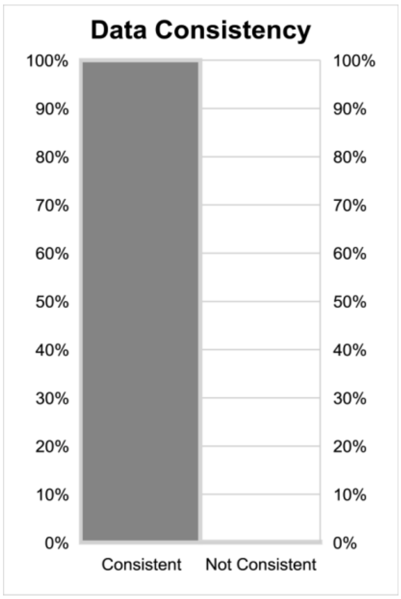


Figure 5-7: Percentage of consistent/inconsistent data after a system crash

Therefore, data availability and its consistency was always preserved during experiments in testbed.

5.4 Evaluation

This section introduces a discussion about the results of the testbed that helps to understand whether or not the defined HA objectives were met by prototype implementation based on reference architecture.

5.4.1 Performance Evaluation

Based on results from the testbed, an overall performance of the prototype cluster can be described as good. Handling of basic “fail-safe” scenarios was as expected.

However, one drawback was discovered during testbed experiments. The switchover process between active and passive nodes is not a straightforward as desired. Active/passive is one of the possible HA configurations, and both, Pacemaker and Corosync, were designed to work in multi-mode cluster environments. Therefore, there is no easy way to conduct a switchover (i.e. using one command or well-documented approach) in the active/passive cluster. Pacemaker provides a possibility to put an active node offline and in that way cause a migration of all cluster resources to the other node. However, in this scenario the master/slave set of DRBD resources will be deactivated on the “offline” node. As a side effect, data replication will be broken and the “offline” node (that is still running) will not be able to take over cluster resources in case of failover.

A possibility for group cluster resources and to create colocation and order constraints between these groups can be used as possible workaround. By doing so, migration of one group will cause a switchover of all cluster groups under the same collocation constraint. Order constraint will be used by the cluster resource manager to start cluster resources in specific order:

- promote replicated block devices on both nodes (active node as Master, passive node as Slave)
- mount on the top of replicated block device file systems
- start the rest of the cluster resources

Therefore, manual switchover scenarios are supposed to be scripted.

5.4.2 Fault Resilience

First, the prototype cluster showed a resilience to crash of the investigated logistics system. Cluster resource manager detected that the process related to cluster resources were not running anymore and automatically started it. It also proved a resilience to hazardous "split-brain" situations. During a communication break one node was powered off (i.e. put in "known state") to prevent any possible data corruptions. The other node stayed online and could continue running cluster resources.

The prototype cluster was fault tolerant to hard drive disk failure and network link failure. Such performance might reduce a number of outages of the investigated logistics system under real-world deployment (for details see Subsection 4.1.2).

Finally, it was an exceptional result that the prototype cluster could continue operating after detaching all local hard drives from the active node. Software-defined storage just transparently fell back in diskless-mode and routed all read/writes to the passive node. The partly destroyed node continued to run the investigated logistics system that could still able coordinate the warehouse actives. Results from data protection experiment (for details see Subsection 5.3.5) proved that no data would be lost in this extreme scenario.

5.4.3 Failover Performance

According to Schmidt (2006, p. 166), automatic failover success of less than 75% can be defined as a bad implemented cluster and more than 90% of successful failovers is a sign of a good cluster. Hence, achieved automatic failover success rate of 100% is remarkable and validates prototype implementation as a well-designed cluster. Moreover, average failover time has a direct correlation to two defined HA objectives for a logistics system: RTO for the logistics system and its MTTR.

However, testbed observations uncovered a pitfall of low MTTR for such complex interdependent systems as warehouse automation systems. Every failover introduced additional "ramp-up" time until warehouse operation was fully restored. Ramp-up time included IT-related issues caused by loss of state due to failover (e.g. stale TCP connections, unfinished transaction, memory-mapped files with PLC status information were not flushed back to disk, etc.) and non-IT related issues that required human intervention in warehouse (e.g. trays with unknown status, powering on warehouse equipment after emergency shutdown, resending unfinished orders). Therefore, for the investigated logistics system higher MTBF (i.e. uptime) is preferable over lower MTTR (i.e. fast failover).

5.4.4 Data Protection

Both “zero data loss” and assured data consistency, resulting in the highest level of data protection, were achieved in the prototype cluster using synchronous replication of data between the nodes without deploying any external storage system. Results from Subsection 5.3.5 proved the fact that no write operation was complete unless it was completed on both nodes. Applications waited for the transaction to be replicated on both servers before moving on. Such blocking behavior had insignificant performance overhead but it ensured that no data was lost when the node fell out. Performance overhead can be further reduced by using RDMA technologies ensuring minimal latencies (see Subsection 3.1.7 for detailed explanation).

Nevertheless, one limitation of proposed approach should be mentioned. The experiment was conducted on a test file that was opened using `O_DIRECT` and `O_SYNC` flags (cf. (Love, 2013, p. 74)). These were the default options used by the investigated logistics system to be sure that data would be synchronously written to the stable storage bypassing operating system page cache. Using other flags to open a file and begin to write to it may produce other results during system crash.

Finally, such approach of data protection can be combined with off-site disaster recovery solution to achieve “Tier 6” solution for business continuity (for details see Table 3-4), where a choice between synchronous and asynchronous replication modes would be still based on performance/latency trade-off.

6 FINDINGS

To be able to understand the results presented in Chapter 5 an analysis is done to summarize the most important findings.

The following findings found support in the experimental evidence:

- The designed prototype can be classified as high available clustered solution (for details see Table 2-3 or Table 2-4 and Section 5.4).
- The investigated logistics system can be running on the prototype cluster without any modifications (for details see Section 5.1).
- The implemented prototype cluster ensures zero data loss in case of a node failure (i.e. total destruction of node) and meets the defined RPO of 0 by introducing synchronous data replication over network between two nodes at block level, the lowest abstraction level available for operating system to access traditional block-level storage (for details see Table 4-1 and Subsection 5.4.4).
- The prototype cluster provides manual switchover and automatic failover (for details see Subsections 5.3.1 and 5.3.4).
- Average failover time for the investigated logistics system was 147.13 seconds, varying from 100 to 198 seconds. Such results exceed the defined RTO of 5 minutes for the investigated logistics system (for details see Table 4-1 and Subsection 5.3.3).
- In case of failover dependent stateful subsystems need additional time to normalize their operations. An overall ramp-up time for fully restored automated operations in warehouse is slowed down by many external factors, including “stuck” TCP connections, “missing” trays on a conveyor belt, start of warehouse equipment after emergency shutdown, etc. Thus, for the investigated logistics system prolonged uptime (MTBF) is preferable over fast failover time (MTTR) (for details see Subsection 5.4.3).
- Latency is a constraining technical factor that limits the proposed approach to local deployments directly at warehouse sites. Both, real-time activities of equipment controllers directed by warehouse control systems and performance overhead due to blocking a nature of synchronous replication require the lowest possible network latency (for details see Subsections 4.1.3 and 5.4.4).
- The prototype cluster was shown to be fault-resilient and fault-tolerant to a number of hardware and software failures (for details see Table 5-10 in Section 5.3).
- In terms of CAP theorem, the prototype cluster is a CP distributed system when both nodes are up and receive “heartbeats” from each other or if one node cleanly reported as being down (i.e. there is no partition) (for details see Subsection 5.3.2).
- Node isolation using fencing lights-out devices is a feasible technique to overcome “split-brain” scenarios and to prevent data loss or data incontinences in two-node cluster setups

when nodes lose their connection to each other (i.e. partitioned) (for details see Subsection 4.2.7 and Subsection 5.3.2).

- The prototype cluster is based exclusively on free and open-source software that runs on commodity server hardware (for details see Subsection 5.1). Both, RAS features of commodity server hardware and FOSS clustering software deliver an acceptable level of availability for a logistics system (for details see Section 5.3).
- Studiously avoided unnecessary complexity delivers simple and reliable solution (for details see Sections 4.2 and 5.3).

7 CONCLUSION

The research question that this study seeks to answer is: "*How can availability of a logistics system be improved at application- and platform-layers, while reducing costs at infrastructure-layer?*" All parts of the question were addressed and answered in the course of the master's thesis, established hypotheses found support in the findings from the experimental setup.

First, the initially proposed theoretical approach to improve availability of the logistics system by utilizing computer clustering concepts was subsequently proved to be a feasible practical solution to attain high availability in the logistics setting of warehouse automation systems. Furthermore, introduced N+1 redundancy to hardware, adopted shared-nothing architecture with synchronous data replication combined with "state of the art" software implementations are shown as effective methods to deal with the problem statement. Finally, to pursue efficiency and cost-effectiveness the proposed solution, in form of two-node HA cluster, was intended to be deployed using free and open-source software running on off-the-shelf commodity servers.

7.1 Outlook and Future Work

First, the proposed clustering approach to improve availability is of practical significance for KNAPP and companies that are seeking cost-effective high availability solutions that provide continuous uptime for their business-critical logistics systems. Such an approach may be considered an industry blueprint to improve availability of cluster-unaware or legacy logistics systems without need to re-architect them.

Nevertheless, some limitations of the thesis should be mentioned. Prototype implementation alone should not be viewed as a "one size fits for all" solution. The outlined reference architecture of two-node HA cluster implies a mix of various technologies. Consequently, independent technical assessment may provide an unbiased discrete implementation to meet business and technical requirements at its best. Such spin-off implementations could include tactics for transaction-level data replication using facilities of underlying database system instead of or as a supplement option to the proposed ubiquitous block-level data replication.

Extending the outlined design with virtualization as an extra layer, which is dependent on the physical infrastructure, may be seen as an opportunity to run practically any logistics system. However, there could be a risk to introduce unnecessary complicity and potential performance penalty that can negatively impact the availability of such systems. Therefore, it might be necessary to conduct experimental evaluations of such implementations in logistics setting.

Since the proposed prototype implementation lacks a transparent failover for a logistics system, another possible research direction is seen in evaluation of virtualization-enabled live migration approach for critical systems as alternative for a failover high availability clustering. Such a concept makes it possible to restore a previous program state of a logistics system upon switchover from a failed active node to the standby node in a seamless way.

Moreover, maintainability of the proposed prototype cluster in the long run might be an interesting subject. Therefore, such topics as automated deployment and orchestration deserve additional study.

Further, reference architecture could be improved to achieve better horizontal scalability by introducing additional nodes in order to overcome possible performance issues if a logistics system grows or to handle burst loads (e.g. during Black Friday and Cyber Monday sales). On the other hand, this may imply certain adjustments of cluster-unware applications and may have a huge impact on the related process of software engineering and software development.

Finally, the proposed reference architecture may be extended to include off-site disaster recovery capabilities in order to support business continuity in a more advanced way.

APPENDIX A -Anaconda Kickstart File

```
#version=TESTLAB
# System authorization information
auth --enableshadow --passalgo=sha512
# Install OS instead or upgrade
install
repo --name="Server-HighAvailability" --baseurl=file:///run/in-
stall/repo/addons/HighAvailability
# Use text mode install
text
# Firewall configuration
firewall --disabled
# Run the Setup Agent on first boot
firstboot --reconfig
ignoredisk --only-use=sda
# Keyboard layouts
keyboard --vckeymap=us --xlayouts='us'
# System language
lang en_US.UTF-8
# Network information
network  --bootproto=dhcp
network  --bootproto=dhcp --hostname=testlab
# Reboot after installation
reboot
# Root password
rootpw --plaintext ch4n63m3
# SELinux configuration
selinux --disabled
# Do not configure the X Window System
skipx
# System timezone
timezone Europe/Vienna --isUtc
# System bootloader configuration
bootloader --append="vconsole.keymap=us vconsole.font=latarcyrheb-sun16
vga=791 crashkernel=auto" --location=mbr --driveorder="sda" --boot-
drive=sda
# Clear the Master Boot Record
zerombr
# Partition clearing information
clearpart --all --initlabel
# Disk partitioning information
part /boot --fstype="ext4" --ondisk=sda --size=512 --label=boot
part /boot/efi --fstype="efi" --ondisk=sda --size=200 --fsoptions="de-
faults,uid=0,gid=0,umask=0077,shortname=winnt" --label=bootefi
part pv.66 --fstype="lvmpv" --ondisk=sda --size=152887
```

```
volgroup rootvg --pesize=4096 pv.66
logvol /  --fstype="xfs" --size=25600 --name=root --vgname=rootvg
logvol swap  --fstype="swap" --size=16384 --name=swap --vgname=rootvg
logvol /home  --fstype="xfs" --size=5120 --name=home --vgname=rootvg
logvol /opt  --fstype="xfs" --size=5120 --name=opt --vgname=rootvg
logvol /var  --fstype="xfs" --size=15360 --name=var --vgname=rootvg
logvol /tmp  --fstype="xfs" --size=5120 --name=tmp --vgname=rootvg
logvol /kisoft/wcs  --fstype="xfs" --size=51200 --name=kisoft-wcs --
vgname=extvg
logvol /kisoft/dbdata  --fstype="xfs" --size=51200 --name=kisoft-dbdata --
vgname=extvg

%post --logfile /root/ks-post.log
logger "Starting Anaconda postinstall section"
# Redirect stdout to VT3 and switch to it
exec < /dev/tty3 > /dev/tty3
/usr/bin/chvt 3
set -x -v

echo "Starting post-installation script at $(date +"%m-%d-%Y %T")"

# Adjust /etc/fstab entries
# /dev/mapper/VGNAME-LVNAME -> /dev/VGNAME/LVNAME
sed -i -e '/^\/dev\/mapper\//s/\/mapper//g' -e 's/--/-/g' -e '/vg-/s/-
/\//' /etc/fstab

# /dev/sdXY instead of UUID
for uuid in $(awk '/^UUID=/ {print $1}' /etc/fstab | sed -e 's/UUID=//g' -
e 's/\"//g'); do
        dev=$(blkid -U "$uuid")
        sed -i s#UUID="$uuid"#"$dev"# /etc/fstab
done

echo "Executing of post-installation script finished at $(date +"%m-%d-%Y
%T")"

exit 0
%end

%packages
@^infrastructure-server-environment
@base
@compat-libraries
@core
@debugging
@development
@ha
```

```
@hardware-monitoring
@network-tools
@performance
@perl-runtime
@system-management
%end
```

Listing 7-1: Anaconda Kickstart file

APPENDIX B -HA Cluster Setup

```
#!/bin/bash

# Cluster IP
VIP="172.16.0.1"
NETMASK="255.255.255.0"
NETMASK_CIDR=$(mask2cdr $NETMASK)

# LOM credentials for fencing
LOM_USER="lom_user"
LOM_PASSWD="ch4n63m3"

HA_PASSWD="ch4n63m3"

CURHN=$(hostname -s)

if [[ $CURHN =~ ^[A-Za-z0-9-]{2,}[1]$ ]]; then
  N1="$CURHN"
  N2="${CURHN::-1}2"
  NODE="node-1"
  PEER="node-2"
else
  N1="${CURHN::-1}1"
  N2="$CURHN"
  NODE="node-2"
  PEER="node-1"
fi

# String manipulations for cluster name:
CL_NAME=${CURHN,,}
CL_NAME=${CL_NAME/-srv/}
CL_NAME=${CL_NAME::-1}

is_master() {
  if [[ $CURHN =~ ^[A-Za-z0-9-]{2,}[1]$ ]]; then
    return 0
  else
    return 1
  fi
}

mask2cdr() {
  # Assumes there's no "255." after a non-255 byte in the mask
  local x=${1##*255.}
  set -- 0^^^128^192^224^240^248^252^254^ $(( (${#1} - ${#x})*2 )) ${x%.*}
  x=${1%$3*}
  echo $(( $2 + (${#x}/4) ))
}

sync_nodes() {
  echo "Waiting for nodes $N1 and $N2 to come up..."
  until [[ $(nmap -n -p 2224 $N1 $N2 2>/dev/null | grep -c "open") -eq 2
]]; do
    sleep 10
  done
  echo "Nodes $N1 and $N2 are up"
}

is_cluster_active() {
  if systemctl -q is-active corosync; then
    CL_NAME=$(pcs status 2> /dev/null | grep "Cluster name:" | cut -d ' '
-f 3)
    if [[ ! -z $CL_NAME ]]; then
      return 0
    else
      return 1
    fi
  else
    return 1
  fi
}

cluster_init() {
  echo "Authenticate cluster nodes with each other"
  pcs cluster auth $N1 $N2 -u hacluster -p $HA_PASSWD --force > /dev/null
2>&1
  sleep 3

  echo "Configure corosync, RRP with 2 rings and sync configuration out to
both nodes"
  pcs cluster setup --name $CL_NAME $N1,node-1 $N2,node-2 --token 5000 --
join 60 --consensus 6000 --force > /dev/null 2>&1
  sleep 3

  echo "Starting HA cluster"
  pcs cluster start --all > /dev/null 2>&1
  sleep 3

  echo "Configuring cluster properties"
  pcs property set no-quorum-policy=ignore
  pcs property set stonith-enabled=false
  pcs property set dc-deadtime=60s
  pcs property set batch-limit=30
  pcs property set default-action-timeout=600s
  pcs property set default-resource-stickiness=100
  pcs property set start-failure-is-fatal=false
  pcs property set pe-error-series-max=10
  pcs property set pe-warn-series-max=10
  pcs property set pe-input-series-max=10
}

cluster_create_resources() {
  echo "Adding cluster IP"
  pcs resource create p_vip_knapp ocf:heartbeat:IPaddr2 ip=$VIP cidr_net-
mask=$NETMASK_CIDR iflabel=0 op monitor interval=10s --group=g_cluster_res

  echo "Adding basic resource primitives and assign them to groups"
  # Create cluster resources
  pcs resource create p_oracledb systemd:oracledb op monitor interval=15s
  pcs resource group add g_cluster_res p_oracledb
  pcs resource create p_kisoft_wcs systemd:wcs --group=g_cluster_res

  # Create order and colocation constraints
  pcs constraint order g_cluster_res then start g_wcs id=o_g_wcs_af-
ter_g_cluster_res
  pcs constraint colocation add g_wcs with g_cluster_res score=INFINITY
id=c_g_wcs_on_g_cluster_res

  # Fencing
  fence_delay="5"
  for node in $(crm_node -l | awk '{print $2}'); do
    lom_ip=$(ssh root@$node "ipmitool lan print | grep "IP Address  " |
awk -F': ' '{print $2}'")
    pcs stonith create fence_$node fence_ilo pcmk_host_list="$node"
ipaddr="$lom_ip" login="$LOM_USER" passwd="$LOM_PASSWD" delay=$fence_delay
action="off"
    fence_delay=$((fence_delay+1))
  done
}

cluster_finalize() {
  echo "Configuring corosync & pacemaker to run on node boot"
  pcs cluster enable --all

  echo "Resetting failure counters"
  pcs resource cleanup

  echo "Verifying cluster configuration"
  pcs cluster verify || return 1
```

```
}

install_ssh_keys() {
  ssh-keygen -f /root/.ssh/id_rsa -t rsa -N ''

  expect -c "
set timeout 1200;
spawn ssh-copy-id root@$PEER
expect {
  \"*yes/no*\" {send \"yes\r\"; exp_continue}
  \"*password*\" {send \"$HA_PASSWD\r\";}
}
expect eof;"
}

echo "Setting password for hacluster user"
echo "$HA_PASSWD" | passwd -f --stdin hacluster

echo "Starting pacemaker configuration daemon"
systemctl -q daemon-reload
systemctl -q is-active pcsd || systemctl -q enable pcsd && systemctl -q
start pcsd

sync_nodes
install_ssh_keys

echo "Setting up HA cluster..."

if is_master; then
  echo "We are a active-node. Continue..."
  cluster_init
  cluster_create_resources
  if ! cluster_finalize; then
    echo "Syntax or conceptual error in pacemaker configuration"
  fi

else
  echo "We are a slave-node. Nothing to do."
fi

echo "HA cluster setup is finished."

exit 0
```

Listing 7-2: Script for HA cluster deployment

APPENDIX C - Resilient Storage Setup

```
#!/bin/bash

# Disable the printing of messages to the console
dmesg -D

# Logical volumes for DRBD devices
LV_DRBD="kisoft-wcs kisoft-dbdata"

DRBD_MINOR_NR=0
DRBD_PORT=7788

CURHN=$(hostname -s)

if [[ $CURHN =~ ^[A-Za-z0-9-]{2,}[1]$ ]]; then
  N1="$CURHN"
  N2="${CURHN::-1}2"
else
  N1="${CURHN::-1}1"
  N2="$CURHN"
fi

# Cluster interlink addresses for nodes
N1IP="192.168.0.1"
N2IP="192.168.0.2"

# Function to generate a DRBD resource file
#
# arguments
# $1 LV name
# $2 VG name
# $3 node 1 hostname
# $4 node 1 ip
# $5 node 2 hostname
# $6 node 2 ip
create_drbd_res() {
  local lvname=$1
  local vgname=$2
  local n1name=$3
  local n1ip=$4
  local n2name=$5
  local n2ip=$6

  echo "resource $lvname {
```

```
                device                                  /dev/drbd$DRBD_MI-
NOR_NR;
                disk
/dev/${vgname}/$lvname;
                flexible-meta-disk                      internal;
                on $n1name {
                                address
$n1ip:$DRBD_PORT;
                }
                on $n2name {
                                address
$n2ip:$DRBD_PORT;
                }
  }" > /etc/drbd.d/${lvname}.res

}

sync_nodes() {
  echo "Waiting for nodes $N1 and $N2 to come up..."
  until [[ $(nmap -n -p 2224 $N1 $N2 2>/dev/null | grep -c "open") -eq 2
]]; do
    sleep 10
  done
  echo "Nodes $N1 and $N2 are up"
}

is_master() {
  if [[ $CURHN =~ ^[A-Za-z0-9-]{2,}[1]$ ]]; then
    return 0
  else
    return 1
  fi
}

lvm_setup() {
  # Set up the LVM environment
  test -e /etc/lvm/lvm.conf && cp /etc/lvm/lvm.conf /etc/lvm/lvm.conf.orig
  # Disable LVM cache
  sed -i 's/write_cache_state = 1/write_cache_state = 0/g'
/etc/lvm/lvm.conf
  rm -rf /etc/lvm/.cache
}

drbd_create_config() {
  echo "Generating configuration for DRBD resources"
  for l in $LV_DRBD; do
```

```
    lvg=$(lvs -o lv_name,vg_name --separator " " 2>/dev/null | grep "$l "
| cut -d " " -f4)
    create_drbd_res $l $lvg "$N1" $N1IP "$N2" $N2IP

    DRBD_MINOR_NR=$((DRBD_MINOR_NR+1))
    DRBD_PORT=$((DRBD_PORT+1))
  done
}

drbd_create_res() {
  echo "Creating DRBD resources"
  # Wipe old metadata if any
  for res in $(drbdadm sh-resources 2>/dev/null); do
    resdir="/$(echo $res | sed 's,-,/,g')"
    grep -q "$resdir " /proc/mounts && umount -l $resdir
    dd if=/dev/zero of=$(drbdadm sh-md-dev $res 2>/dev/null) bs=4k count=1
> /dev/null 2>&1
  done
  drbdadm -- --force create-md all
  # Redirect drbd error messages to syslog instead of stderr
  modprobe -s drbd
  drbdadm up all
}

drbd_create_fs() {
  if is_master; then
    echo "Starting the initial full synchronization"
    drbdadm primary --force all
  fi
  for res in $(drbdadm sh-resources 2>/dev/null); do
    resdir="/$(echo $res | sed 's,-,/,g')"
    lbl=$res
    lbl="${lbl//kisoft/}"
    lbl="${lbl//-/}"
    if is_master; then
      echo "Creating file system for $lbl"
      mkfs.xfs -f -q -L $lbl /dev/drbd/by-res/$res/0
    fi
    # Replace /etc/fstab LV entiries with DRBD
    lvg=$(lvs -o lv_name,vg_name --separator " " 2>/dev/null | grep "$res
" | cut -d " " -f4)
    sed -i "s/\/dev\/$lvg\/$res[ \t]/\/dev\/drbd\/by-res\/$res\/0\t/g"
/etc/fstab
    # Append noauto for DRBD mountpoints
    sed -i "/\/dev\/drbd/s/defaults[ \t]/defaults,noauto\t/g" /etc/fstab
  done
}
```

```
drbd2cluster() {
  systemctl -q is-active pcsd || systemctl -q start pcsd
  echo "Adding DRBD to cluster"
  i=0
  for res in $(drbdadm sh-resources 2>/dev/null); do
    if [[ i -eq 0 ]]; then
      pcs resource create p_drbd_$res ocf:linbit:drbd params drbd_re-
source="$res"
      pcs resource group add g_drbd p_drbd_$res
      i=1
    else
      pcs resource create p_drbd_$res ocf:linbit:drbd params drbd_re-
source="$res" --group=g_drbd
    fi
  done

  pcs resource master ms_drbd g_drbd master-max=1 master-node-max=1 clone-
max=2 clone-node-max=1 notify=true

  # Create file system resources on the top of DRBD block devices
  for res in $(drbdadm sh-resources 2>/dev/null); do
    resdir="/$(echo $res | sed 's,-,/,g')"
    pcs resource create p_fs_$res ocf:heartbeat:Filesystem de-
vice="/dev/drbd/by-res/$res/0" directory="$resdir" fstype="xfs" op-
tions="defaults" --group=g_cluster_res --before p_vip_knapp
  done

  pcs constraint order promote ms_drbd then start g_cluster_res
id=o_g_cluster_res_after_ms_drbd
  pcs constraint colocation add g_cluster_res with master ms_drbd
score=INFINITY id=c_g_cluster_res_on_ms_drbd

  pcs resource manage g_drbd
  pcs resource cleanup
}

sync_nodes
lvm_setup

echo "Unloading DRBD"
if systemctl -q is-active drbd || [ -e /proc/drbd ]; then
  systemctl -q stop drbd
  modprobe -r drbd > /dev/null 2>&1
fi

drbd_create_config
```

```
drbd_create_res
drbd_create_fs

if is_master; then
  drbd2cluster

  # Create directories for mount points
  for d in wcs dbdata; do
    until grep -q "drbd.*/kisoft/$d " /proc/mounts; do
      sleep 1
    done
    test -d /kisoft/$d && chmod 777 /kisoft/$d
  done
fi

# Disabling DRBD service (will be managed with cluster)
systemctl -q is-enabled drbd && systemctl -q disable drbd

exit 0
```

Listing 7-3: Script to deploy resilient storage

ABBREVIATIONS

2PC	two-phase commit protocol
ACID	Atomicity, Consistency, Isolation, Durability
B2B	business-to-business
BASE	Basically Available, Soft state, and Eventual consistency
BIA	business impact analysis
CPU	central processing unit
DNS	Domain Name System
DR	disaster recovery
DRaaS	Disaster Recovery as a Service
DRBD	Distributed Replicated Block Device
FOSS	Free and open-source software
ECC	error-correcting code
EDI	Electronic Data Interchange
ERP	enterprise resource planning
HA	high availability
HVAC	Heating, ventilation and air conditioning
GigE	Gigabit Ethernet
I/O	input/output
IPMI	Intelligent Platform Management Interface
IT	information technology
ITIL	Information Technology Infrastructure Library
KI	key indicator
LOM	lights-out management card
MCA	Machine Check Architecture
MTBF	mean time between failures
MTTD	mean time to detection
MTTF	mean time to failure
MTTR	mean time to repair
NIC	network interface controller

NTP	network time server
PLC	programmable logic controller
RAID	redundant array of independent disks
RAM	random-access memory
RAS	reliability, availability, and serviceability
RDMA	remote direct memory access
RFID	radio-frequency identification
RISC	reduced instruction set computing
RPO	recovery point objective
RTO	recovery time objective
RTT	round-trip time
SAN	storage area network
SDN	software-defined network
SDS	software-defined storage
SDx	software-defined anything
SLA	service-level agreement
SPoF	single point of failure
TCP	Transmission Control Protocol
TOGAF	The Open Group Architectural Framework
UDP	User Datagram Protocol
WAN	wide area network
WCS	warehouse control system
WMS	warehouse management system
VM	virtual machine
VMM	Virtual Machine Manager

LIST OF FIGURES

LIST OF TABLES

LISTINGS

LIST OF EQUATIONS

BIBLIOGRAPHY

Adair, R. J., Bayles, R. U., Comeau, L. W., & Creasy, R. J. (1966, May). A Virtual Machine System for the 360/40. *IBM Scientific Center Report No. G320-2007*.

Algirdas, A., Laprie, J.-C., Randell, B., & Landwehr, C. (2004, Jan.-March). Basic Concepts and Taxonomy of Dependable and Secure Computing. *Carl Landwehr IEEE Transactions on Dependable and Secure Computing, 1*(1), 11-33.

Allspaw, J., & Robbins, J. (2010). *Web Operations: Keeping the Data on Time.* O'Reilly.

Antony, B., Boudnik, K., Adams, C., Shao, B., Lee, C., & Sasaki, K. (2016). *Professional Hadoop.* Wiley.

Atchison, L. (2016). *Architecting for Scale: High Availability for Your Growing Applications.* O'Reilly.

AXELOS. (2011). *ITIL® Glossary and Abbreviations.* Retrieved from https://www.axelos.com/Corporate/media/Files/Glossaries/ITIL_2011_Glossary_GB-v1-0.pdf

Bach, M. (2014). *Expert Consolidation in Oracle Database 12c.* Apress.

Backblaze. (2017, May 09). *Hard Drive Stats for Q1 2017*. Retrieved from Hard Drive Test Data: https://www.backblaze.com/blog/hard-drive-failure-rates-q1-2017/

Bairavasundaram, L. N., Goodson, G. R., & Schroeder, B. (2008). An Analysis of Data Corruption in the Storage Stack. *USENIX conference on File and Storage Technologies*, (pp. 223–238).

Bakesa, C. M., Kimb, C. M., & Ramosb, C. T. (2003). An assessment of Gigabit Ethernet technology and its applications at the NASA Glenn Research Center: a case study. *Engineering and Technology Management*, 245-272.

Bauer, E., & Adams, R. (2012). *Reliability and Availability of Cloud Computing.* Wiley.

Bauer, E., Randee, A., & Eustace, D. (2011). *Beyond Redundancy: How Geographic Redundancy Can Improve Service Availability and Reliability of Computer-Based Systems.* Hoboken, New Jersey: Wiley.

Beekhof, A. (2015). *Clusters from Scratch.* Retrieved from ClusterLabs: http://clusterlabs.org/doc/Cluster_from_Scratch.pdf

Berkowitz, H. (2002). *Building Service Provider Networks.* Wiley.

Bernadette Charron-Bost, F. P. (2010). *Replication: Theory and Practice.* Springer.

Bernstein, P. A., Hadzilacos, V., & Goodman, N. (1987). *Concurrency Control and Recovery in Database Systems.* Addison-Wesley.

Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (2016). *Site Reliability Engineering: How Google Runs Production Systems.* O'Reilly.

Bressoud, T. C., & Schneider, F. B. (1996). Hypervisor-based Fault-tolerance. *ACM Transactions on Computer Systems, 14*(1), 90-107.

Bundesministeriums für Verkehr, Innovation und Technologie. (2015). *Industrie 4.0 und ihre Auswirkungen auf die Transportlogistik*. Retrieved from https://www.bmvit.gv.at/innovation/publikationen/verkehrstechnologie/downloads/industrie_4_0.pdf

Buschmann, F., Henney, K., & Schmidt, . C. (2007). *Pattern-Oriented Software Architecture, A Pattern Language for Distributed Computing* (Vol. 4). Wiley.

Business Continuity Institute. (2011, September). *Dictionary of Business Continuity Management Terms.* Retrieved from http://www.thebci.org/glossary.pdf

Business Continuity Institute. (2015, November). *Supply Chain Resilience Report 2015.* Retrieved from http://www.bcifiles.com/bci-supply-chain-resilience-2015.pdf

Buyya, R. (1999). *High Performance Cluster Computing* (Vol. 1). Upper Saddle River, NJ: Prentice Hall.

CA Technologies. (2011, January). *The Avoidable Cost of of Downtime.* Retrieved from http://www.ca.com/~/media/files/articles/avoidable_cost_of_downtime_part_2_ita.aspx

Calzolari, F., Arezzini, S., Ciampa, A., Mazzoni, E., Domenici, A., & Vaglini, G. (2010). High Availability using Virtualization. *17th International Conference on Computing in High Energy and Nuclear Physics*, (pp. 1-100).

Carlson, M., Yoder, A., Schoeb, L., Deel, D., & Pratt, C. (2014, April). *Software Defined Storage.* Retrieved from http://www.snia.org/sites/default/files/SNIA%20Software%20Defined%20Storage%20White%20Paper-%20v1.0k-DRAFT.pdf

Casey, M. (2016, January 08). *Announcing the general availability of Unbreakable Enterprise Kernel Release 4*. Retrieved from Oracle's Linux Blog: https://blogs.oracle.com/linux/announcing-the-general-availability-of-unbreakable-enterprise-kernel-release-4

Castano, V., & Schagaev, I. (2015). *Resilient Computer System Design.* Springer.

Caulfield, C. (2016, January). *New Quorum Features in Corosync 2.0.* Retrieved from http://people.redhat.com/ccaulfie/docs/Votequorum_Intro.pdf

CeMAT. (2016a, April 11). *CeMAT 2016: Sharp focus on IT. Press Release*. Retrieved from http://www.cemat.de/files/005-fs5/media/downloads/journalists/2016/deutsche-messe-cemat-2016-sharp-focus-on-it.pdf

CeMAT. (2016b, May 17). *CeMAT 2016: CeMAT to be co-staged with HANNOVER MESSE starting in 2018. Press Release.* Retrieved from http://www.cemat.de/files/005-fs5/media/downloads/journalisten/2016/047-2016-en-cemat-2018-vg.rtf

Chen, P. M., Lee, E. K., Gibson, G. A., Katz, R. H., & Patterson, D. A. (1994). RAID: High-Performance, Reliable Secondary Storage. *ACM Computing Surveys, 26*(2).

Ciarfella, P., Moser, L., Melliar-Smith, P., & Agarwal, D. (1994). The Totem Protocol Development Environment. *International Conference on Network Protocols*, (pp. 168-177). Boston, MA.

Colman-Meixner, C., Develder, C., Tornatore, M., & Mukherjee, B. (2016, February 18). A Survey on Resiliency Techniques in Cloud Computing Infrastructures and Applications. *IEEE Communications Surveys & Tutorials*, 1-38.

Continuity Software. (2014, May). *2014 Service Availability Benchmark Survey.* Retrieved from http://www.continuitysoftware.com/wp-content/uploads/2014/05/2014-SA-Survey-Report.pdf

Critchley, T. (2015). *High Availability IT Services.* Boca Raton: CRC Press.

Cully, B., Lefebvre, G., Meyer, D., Feeley, M., Hutchinson, N., & Warfield, A. (2008). Remus: High Availability via Asynchronous Virtual Machine Replication. *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, (pp. 161-174).

Das, S., Agrawal, D., & Abbadi, A. E. (2010). G-Store: A Scalable Data Store for Transactional Multi key Access in the Cloud. *1st ACM Symposium on Cloud Computing* (pp. 163-174). New York: ACM.

DELL. (2016, March). *5 Ways to Ensure Reliability, Availability, and Serviceability in Your Enterprise Environment.* Retrieved from http://i.dell.com/sites/doccontent/shared-content/data-sheets/en/Documents/Dell-PowerEdge-R930-RAS-Whitepaper.pdf

Deloitte. (2015). *Industry 4.0 – Challenges and solutions for the digital transformation and use of exponential technologies.* Retrieved from http://www2.deloitte.com/content/dam/Deloitte/ch/Documents/manufacturing/ch-en-manufacturing-industry-4-0-24102014.pdf

Disaster Recovery Journal. (2014). *The State of IT Resiliency and Preparedness.* Retrieved from http://www.drj.com/images/surveys_pdf/forrester/2013-Forrester-Survey.pdf

Disaster Recovery Preparedness Council. (2014, January). *Disaster Recovery Preparedness Benchmark Survey: 2014 Annual Report.* Retrieved from https://drbenchmark.org/wp-content/uploads/2014/02/ANNUAL_REPORT-DRPBenchmark_Survey_Results_2014_report.pdf

Dyke, J., Shaw, S., & Bach, M. (2011). *Pro Oracle Database 11g RAC on Linux.* Apress.

Elerath, J. G., & Shah, S. (2004). Server class disk drives: how reliable are they? *Annual Symposium on Reliability and Maintainability* (pp. 151-156). IEEE.

Ellenberg, L. (2008). *DRBD 9 - Linux Storage Replication.* Retrieved from http://data.guug.de/slides/lk2008/le_drbd9-lk2008-slides.pdf

Ellenberg, L. (2008). DRBD® 9 & Device-Mapper Linux® Block Level Storage Replication. *LinuxKongress 2008: 15th International Linux System Technology Conference* (pp. 1-12). Hamburg: University of Hamburg.

EMC. (2014). *EMC Global Data Protection Index.* Retrieved from http://www.emc.com/collateral/presentation/emc-dpi-key-findings-global.pdf

Erel, M., Arslan, Z., Yusuf, O., & Canberk, B. (2015). Software-defined wireless network (SDWN). In M. S. Obaidat, F. Zarai, & P. Nicopolitidis (Eds.), *Modeling and Simulation of Computer Networks and Systems: Methodologies and Applications* (pp. 751-766). Morgan Kaufmann.

Ford, D., Labelle, F., Popovici, F., Stokely, M., Truong, V.-A., Barroso, L., . . . Quinlan, S. (2010). Availability in Globally Distributed Storage Systems. *9th USENIX Symposium on Operating Systems Design and Implementation*, (pp. 61-74).

Forrester. (2013, February). *How Organizations are Improving Business Resiliency with Continuous IT Availability.* Retrieved from http://www.emc.com/collateral/analyst-report/forrester-improve-bus-resiliency-continuous-it-avail-ar.pdf

Forrester. (2014). *Building The Always-On, Always-Available Enterprise.* Forrester Research, Inc. Retrieved from https://www.suse.com/docrep/documents/j6eeme6e10/building_alway_on_available_enterprise.pdf

Fox, A., & Brewer, E. (1999). Harvest, Yield, and Scalable Tolerant Systems. *The 7th Workshop on Hot Topics in Operating System* (pp. 174-178). Rio Rico, Arizona: IEEE.

Franke, U. (2012). Optimal IT Service Availability: Shorter Outages, or Fewer? *IEEE Transactions on Network and Service Management, 9*(1), 22-33.

Furmans, K., Nobbe, C., & Schwab, M. (2011). Future of Material Handling – modular, flexible and efficient. *IEEE International Conference on Intelligent Robots and Systems.* San Francisco. Retrieved from http://www.ifl.kit.edu/download/publikationen/Furmans_Nobbe_Schwab-Future_of_material_handling.pdf

Gainaru, A., & Cappello, F. (2015). *Fault-Tolerance Techniques for High-Performance Computing.* (T. Herault, & R. Yves, Eds.) Springer.

Gartner. (2009, December 21). *Competitive Landscape: Clustering Software.* Retrieved from https://www.symantec.com/content/en/us/about/media/industryanalysts/Gartner_Clustering_Software_Market_Dec09.pdf

Gartner. (2013, October 8). *Gartner Identifies the Top 10 Strategic Technology Trends for 2014*. Retrieved from http://www.gartner.com/newsroom/id/2603623

Gilbert, S., & Lynch, N. (2002). Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. *ACM SIGACT News, 33*(2), 51-59.

Gransberg, D. D., Popescu, C. M., & Ryan, R. (2006). *Construction Equipment Management for Engineers, Estimators, and Owners.* CRC Press.

Gray, J., & Siewiorek, D. P. (1991, September). High-Availability Computer Systems. *Computer, 24*(9), pp. 39-48.

Gunasekaran, A. (2007). Developing an E-Logistics System: A Case Study. *International Journal of Logistics: Research and Applications, 10*(4), 333–349.

HA Forum. (2001, February). *Providing Open Architecture High Availability Solutions.* Retrieved from http://docdb.fnal.gov/ILC/DocDB/0000/000084/001/ha-solutions.pdf

Hajinazari, P., & Abbas, A. (2012). E-service Quality Management in B2B e-Commerce Environment. *Third International Conference on Contemporary Issues in Computer and Information Sciences*, (pp. 161-164).

Hale, C. (2010, Oct 07). *You Can't Sacrifice Partition Tolerance*. Retrieved from https://codahale.com/you-cant-sacrifice-partition-tolerance/

Härder, T., & Reuter, A. (1983, December). Principles of Transaction-Oriented Database Recovery. *ACM Computing Surveys, 15*(4), 287-317.

Harris, R. (2012, July 23). *The post-RAID era begins*. Retrieved from http://storagemojo.com/2012/07/23/the-post-raid-era-has-begun/

Harrison, R. (2013). *TOGAF® 9 Foundation Study Guide.* Van Haren Publishing.

Hausladen, I. (2010). Reference Modeling of an IT-Based Logistics System. *8th International Heinz Nixdorf Symposium*, (pp. 234-244). Paderborn.

Hausladen, I. (2016). IT-gestütztes Logistiksystem. In *IT-gestützte Logistik: Systeme - Prozesse - Anwendungen, 3. Auflage* (pp. 29-52). Springer.

Hausladen, I., & Haas, A. (2016). Contribution of IT-Based Logistics Solutions to Sustainable Logistics Management. In *Sustainable Logistics and Strategic Transportation Planning.*

Hawkins, S. M., Yen, D. C., & Chou, D. C. (2000). Disaster Recovery Planning: a Strategy for Data Security. *Information Management & Computer Security, 8*(5), 222-229.

Heavy Reading. (2012, December). *The Virtual Telco: Security & High Availability Are Key to Virtualization Advantage. Whitepaper.* Retrieved from https://www.symantec.com/content/en/us/enterprise/white_papers/b-HR-Symantec-Virtualization-WP.pdf

Heegaard, P. E. (2015). Achieving dependability in software-defined networking – a perspective. *7th International Workshop on Reliable Networks Design and Modeling (RNDM)* (pp. 63-70). Munich: IEEE.

HP. (2013, January). *Selecting the right HP ProLiant scale-up server for your workloads.* Retrieved from http://www8.hp.com/h20195/v2/GetPDF.aspx/4AA4-3475ENW.pdf

HRG. (2003). *Availability Environment Classifications (AEC).* Retrieved from http://www.hrgresearch.com/pdf/AEC%20Defintions.pdf

Hwang, K., Dongarra, J., & Fox, G. C. (2011). *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things.* Elsevier.

IBM. (2012). *z/VM – A Brief Review of Its 40 Year History.* Retrieved from http://www.vm.ibm.com/vm40hist.pdf

IDC. (2013a, November 4). *High Availability: Vital for Mission-Critical Workloads.* Retrieved from http://c.ymcdn.com/sites/www.connect-community.org/resource/resmgr/2013_nonstop_presos/idc_preso_hp_meeting_nov._4_.pdf

IDC. (2013b). *New Potential for High-Availability Solutions Identified from Market Growth and Best Practices: Growth Picks up for Business Continuity, Cloud, and Emerging Economies.* Retrieved from http://www.nec.com/en/global/prod/expresscluster/materials/202991_EN_IDC_NEC_HA_White_Paper.pdf

IDC. (2013c, October). *Where in the World is Storage.* Retrieved from http://www.idc.com/downloads/where_is_storage_infographic_243338.pdf

IEEE. (1990). *IEEE Standard Glossary of Software Engineering Terminology.* (F. Kohn, O. Kutzmutz, & P. Larisch, Eds.) New York: Institute of Electrical and Electronics Engineers.

IEEE. (2002). *IEEE Std 802.3ae-2002, Amendment to IEEE Std 802.3-2002.* New York: IEEE.

IETF. (1989). *RFC 1122: Requirements for Internet Hosts – Communication Layers*. Retrieved from https://tools.ietf.org/html/rfc1122

IETF. (2006). *RFC 4786: Services, Operation of Anycast.* Retrieved from https://tools.ietf.org/html/rfc4786

Intel. (2005). *Reliability, Availability, and Serviceability for the Always-on Enterprise.* Retrieved from http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/reliability-availability-and-serviceability-for-the-always-on-enterprise-paper.pdf

Intel. (2011). *Intel® Xeon® Processor E7 Family for New RAS Servers: White Paper.* Retrieved from http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/xeon-e7-family-ras-server-paper.pdf

Intel. (2012). *Methods to Handle Data Durability Challenges for Big Data.* Retrieved from http://www.intel.eu/content/dam/www/public/us/en/documents/white-papers/big-data-amplidata-storage-paper.pdf

ITIC. (2016, October). *ITIC 2016-2017 Global Server Hardware, Server OS Reliability Report.* Retrieved from http://static.ziftsolutions.com/files/ff80818159fcd304015a1e0fc406693f/ITIC-2016-2017-Global-Server-Hardware-Reliability-Report.pdf

ITIC. (2017, June 14). *ITIC 2017 Global Reliability Survey Mid-Year Update.* Retrieved from www.lenovofiles.com/docs/itic_2017.html

Jewell, D., Dobelin, B. R., Diederichs, S., Duijvestijn, M. L., Hammersley, M., Hazra, A., . . . Stahl, E. (2014). *Performance and Capacity Implications for Big Data.* IBM Redbooks.

Johnson, M. (1992, 10 19). IBM to Allow AS/400 Clustering. *Computerworld, 26*(42), p. 24.

Joy, A. M. (2015). Performance Comparison Between Linux Containers and Virtual Machines. *2015 International Conference on Advances in Computer Engineering and Applications*, (pp. 342-346).

Juve, G., Deelman, E., Vahi, K., Gaurang, M., Berriman, B., Berman, B. P., & Maechling, P. (2009). Scientific workflow applications on Amazon EC2. *5th IEEE International Conference on E-Science Workshops* (pp. 59-66). IEEE.

Kahanwal, B., & Singh, T. P. (2012, July-August). The Distributed Computing Paradigms: P2P, Grid, Cluster, Cloud, and Jungle. *International Journal of Latest Research in Science and Technology, 1*(2), 183-187.

Kanso, A., & Lemieux, Y. (2013). Achieving High Availability at the Application Level in the Cloud. *2013 IEEE 6th International Conference on Cloud Computing* (pp. 778-785). Santa Clara, CA: IEEE Computer Society.

Kaur, J., & Gurm, R. K. (2015, April-June). A Survey on High Availability and Faster Convergence Techniques in IP Networks. *International Journal for Multi-DISciplinary Engineering and Business Management, 3*(2), 59-61.

Kendrick, S. (2012, October). What Takes Us Down? *;login:, 37*(5), 37-45.

Kenneth, B. P. (1987). Exploiting Virtual Synchrony in Distributed Systems. *11th ACM Symposium on Operating Systems Principles* (pp. 123-138). ACM Press.

Kerrisk, M. (2010). *The Linux programming interface: a Linux and UNIX system programming handbook.* San Francisco, CA: No Starch Press.

Khan, O., Burns, R., Plank, J., & Pierce, W. (2012). Rethinking Erasure Codes for Cloud File Systems: Minimizing I/O for Recovery and Degraded Reads. *7th Conference on File and Storage Technologies.* San Jose, CA: USENIX.

Kim, J., Salem, K., Daudjee, K., Aboulnaga, A., & Pan, X. (2015). Database High Availability Using SHADOW Systems. *6th ACM Symposium on Cloud Computing*, (pp. 209-221).

Klappich, D. C. (2013, May 15). *Magic Quadrant for Warehouse Management Systems.* Retrieved from Gartner: https://www.gartner.com/doc/2485715/magic-quadrant-warehouse-management-systems

KNAPP. (2015). *Oracle E-Business Suite Report.* Hart bei Graz.

KNAPP. (2016, July 19). *KNAPP AG balance, fiscal year 2015/16.* Retrieved from https://www.knapp.com/wp-content/uploads/PR_KNAPP_Record_financial_results_for_KNAPP_en-2.pdf

KNAPP. (2017, May). *Philosophies*. Retrieved from KNAPP: https://www.knapp.com/en/solutions/philosophies/

Ko, S. Y., Hoque, I., Cho, B., & Gupta, I. (2010). Making Cloud Intermediate Data Fault-Tolerant. *1st ACM symposium on Cloud computing, SoCC '10* (pp. 181-192). New York: ACM.

Koren, I., & Mani Krishna, C. (2010). *Fault-Tolerant Systems.* Morgan Kaufmann.

KPMG. (2014). *Technology Risk Radar. 2nd Edition.* Retrieved from https://www.kpmg.com/UK/en/IssuesAndInsights/ArticlesPublications/Documents/PDF/Market%20Sector/Technology/tech-risk-radar-second-edition.pdf

Kronenberg, N. P., Levy, H. M., & Strecker, W. D. (1986, May). VAXcluster: a Closely-coupled Distributed System. *ACM Transactions on Computer Systems, 4*(2), 130-146.

Kyne, F., Clifton, A., Deane, J., Ferreira, F., Gunjal, R., Laurent, C., . . . Zemotel, Y. (2014). *Improving Z/Os Application Availability by Managing Planned Outages.* IBM Redbooks.

Laan, S. (2017). *IT Infrastructure Architecture - Infrastructure Building Blocks and Concepts. 3rd Edition.* Lulu Press.

Lee, P. A., & Anderson, T. (1990). *Fault Tolerance: Principles and Practice, 2nd Edition.* Vienna: Springer.

Lehmann, W. (2009). *Linux implementation for the ISP & data center .*

Lenovo. (2016, June). *Lenovo System x3550 M5 Installation and Service Guide.* Retrieved from http://publib.boulder.ibm.com/infocenter/systemx/documentation/topic/com.lenovo.sysx.5463.doc/PDF_5463_isg.pdf

Leukel, J., Ludwig, A., & Norta, A. (2011). Introduction to the Second International Workshop on Service Oriented Computing in Logistics (SOC-LOG 2010). In *Service-Oriented Computing* (pp. 210-212). Springer.

Li, W., Kanso, A., & Gherbi, A. (2015). Leveraging Linux Containers to Achieve High Availability for Cloud Services. *2015 IEEE International Conference on Cloud Engineering (IC2E)* (pp. 76-83). Tempe, AZ: IEEE.

Li, X. (2012). The New Trend of Security in Cloud Computing. *International Journal of Engineering Innovation & Research, 1*(6), 516-519.

Liebel, O. (2013). *Linux Hochverfügbarkeit: Einsatzszenarien und Praxislösungen für Linux-Server.* Bonn: Galileo Press.

LINBIT. (2016). *The DRBD User's Guide.* Retrieved from http://docs.linbit.com/docs/users-guide-8.4/

Linux Foundation. (2011). *Carrier Grade Linux 5.0 Specification.* Retrieved from https://www.linuxfoundation.org/sites/main/files/CGL_5.0_Specification.pdf

Love, R. (2013). *Linux System Programming, 2nd Edition.* O'Reilly.

Lu, M., & Chiueh, T.-c. (2009). Fast Memory State Synchronization for Virtualization-based Fault Tolerance. *39th IEEE International Conference on Dependable Systems and Networks.* Lisbon.

Lumpp, T., Schneider, J., Holtz, J., Mueller, M., Lenz, N., Biazetti, A., & Petersen, D. (2008). From high availability and disaster recovery to business continuity solutions. *IBM Systems Journal*, 605-619.

Maier, M. M. (2011). *Praxisgerechte Abnahmeprozeduren für intralogistische Systeme unter Berücksichtigung der Zuverlässigkeits- und Verfügbarkeitstheorie.* Fakultät für Maschinenbau. Ilmenau: Technischen Universität Ilmenau.

Marcus, E., & Stern, H. (2003). *Blueprints for High Availability, 2nd Edition.* Indianapolis: Wiley Press.

Marshall, N., & Lowe, S. (2015). *Mastering VMware vSphere 6.* Wiley.

Miller, K. (2012, January 9). *Calculating Optical Fiber Latency*. Retrieved from http://www.m2optics.com/blog/bid/70587/Calculating-Optical-Fiber-Latency

Moniruzzaman, A., & Hossain, S. A. (2014, March). A Low Cost Two-tier Architecture Model Implementation for High Availability Clusters For Application Load Balancing. *Journal of Electronic Systems, 4*(1), 25-32.

Morabito, R., Kjällman, J., & Komu, M. (2015). Hypervisors vs. Lightweight Virtualization: a Performance Comparison. *2015 IEEE International Conference on Cloud Engineering (IC2E)*, (pp. 386-393).

Muhamedagic, D. (2016, 11 02). *Fencing and Stonith*. Retrieved from http://clusterlabs.org/doc/crm_fencing.html

Mullins, C. (2002). *Database Administration: The Complete Guide to Practices and Procedures.* Boston: Addison-Wesley.

Murugesan, S., & Bojanova, . (Eds.). (2016). *Encyclopedia of Cloud Computing.* Wiley.

Nadeau, T. D., & Gray, K. (2013). *SDN: Software Defined Networks.* O'Reilly.

NEC. (2016). *Smart Enterprise Drivers.* Retrieved from https://www.necam.com/docs/?id=7bbf6c9f-400c-4c88-ba9d-91736489b390

Nikolaus, K. (2013). Building the Nuts and Bolts of Self-Organizing Factories. *Pictures of the Future*, 19-22.

Obasi, C., Asagba, P., & Silas, A. (2015). A Comparative Study of Consistency Theorems in Distributed Databases. *African Journal of Computing & ICT*, 205-208.

O'Brien, M. (2017, March 21). *Selecting the Right WMS for Your.* Retrieved from Multichannel Merchant: http://multichannelmerchant.com/wp-content/uploads/2017/03/29715_MCM_ExSOperations_WM_v3.pdf

Oggerino, C. (2001). *High Availability Network Fundamentals.* Indianapolis, IN: Cisco Press.

Ohtsuji, H., & Tatebe, O. (2015). Active Storage Mechanism for Cluster-Wide RAID System. *2015 IEEE International Conference on Data Science and Data Intensive Systems* (pp. 25-32). Sydney: IEEE.

Oracle. (2015a, March). *White Paper: Best Practices for Synchronous Redo Transport.* Retrieved from http://www.oracle.com/technetwork/database/availability/sync-2437177.pdf

Oracle. (n.d.). *Oracle Database Runs Best on Oracle Linux*. Retrieved June 22, 2017, from Oracle Technology Network: http://www.oracle.com/technetwork/server-storage/linux/technologies/rdbms-12c-oraclelinux-1973518.html

Orenstein, G. (2003). *IP Storage Networking: Straight to the Core.* Addison-Wesley.

Pall, G. A. (1987). *Quality Process Management.* Prentice-Hall.

Patterson, D., Gibson, G., & Katz, R. H. (1988). A Case for Redundant Arrays of Inexpensive Disks (RAID). *ACM SIGMOD International Conference on Management of Data* (pp. 109-116). Chicago: ACM Press.

Pearl, R. (2015). *Healthy SQL: A Comprehensive Guide to Healthy SQL Server Performance.* Apress.

Perkov, L., Pavković, N., & Petrović, J. (2011). High-availability using open source software. *MIPRO, 2011 Proceedings of the 34th International Convention* (pp. 167-170). Opatija: IEEE.

Perlman, R. (1985). An Algorithm for Distributed Computation of a Spanning Tree in an Extended LAN. *ACM SIGCOMM Computer Communication Review, 15*(4), 44-53.

Piedad, F., & Hawkins, M. (2001). *High Availability: Design, Techniques and Processes.* Upper Saddle River, NJ: Prentice Hall.

Pinheiro, E., Weber, W.-D., & Barroso, L. A. (2007). Failure Trends in a Large Disk Drive Population. *The 5th USENIX Conference on File and Storage Technologies.*

Plank, J. M. (2013, December). Erasure Codes for Storage Systems. A Brief Primer. *;login:, 38*(6), 44-50.

Popek, G. J., & Goldberg, R. P. (1974, July). Formal Requirements for Virtualizable Third Generation Architectures. *Communications of the ACM, 17*(7), 412-421.

Portnoy, M. (2012). *Virtualization Essentials.* Wiley.

Prabhakaran, V., Bairavasundaram, L. N., Agrawal, N., Gunawi, H. S., Arpaci-Dusseau, A. C., & Arpaci-Dusseau, R. H. (2005). IRON File Systems. *The 20th ACM Symposium on Operating Systems Principles.*

Ramabhadran, S., & Pasquale, J. (2006). Analysis of Long-Running Replicated Systems. *25th Annual Joint Conference of the IEEE Computer and Communications Societies*, (pp. 1-9). Barcelona, Spain.

Ramsauer, C. (2013). Industrie 4.0 – Die Produktion der Zukunft. *WINGbusiness 3/2013*, 6-12.

Ray, C. (2009). *Distributed Database System.* Pearson.

Reji, I. (2008). *Logistics Management.* Excel Books.

Resman, M. (2015). *CentOS High Availability.* Packt Publishing.

Rob, P., Coronel, C., & Crockett, K. (2008). *Database Systems: Design, Implementation & Management.* Cengage Learning.

Robinson, H. (2010, April 26). *CAP Confusion: Problems with 'partition tolerance'*. Retrieved from Cloudera Engineering Blog: http://blog.cloudera.com/blog/2010/04/cap-confusion-problems-with-partition-tolerance/

Rogers, P., Janssen, R., Otto, A., Pleus, R., Salla, A., & Sokal, V. (2011). *ABCs of IBM z/OS System Programming* (Vol. 5). IBM Redbooks.

Ronzon, T. (2016, March/April). Software Retrofit in High-Availability Systems: When Uptime Matters. *IEEE Software, 33*(2), 11-17. Retrieved from IEEE Software (Volume:33, Issue: 2): http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=7420474

Salter, J. (2014, January 15). *Bitrot and atomic COWs: Inside "next-gen" filesystems*. Retrieved from http://arstechnica.com/information-technology/2014/01/bitrot-and-atomic-cows-inside-next-gen-filesystems/

Schmidt, K. (2006). *High Availability and Disaster Recovery.* Springer.

Schönig, H.-J. (2015). *PostgreSQL Replication. 2nd Edition.* Packt Publishing.

Schroeder, B., & Gibson, G. A. (2007). Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You? *5th USENIX Conference on File and Storage Technologies* (pp. 1-16). Berkeley, CA: USENIX Association.

Schulz, G. (2017). *Software-Defined Data Infrastructure Essentials: Cloud, Converged, and Virtual Fundamental Server Storage I/O Tradecraft.* Boca Raton: CRC Press.

Schulze, L. (2007). *Redundancy in Warehouses: Technical Constructions, Operation Strategies and their Impact on Throughput.* Planning and Controlling of Warehouse and Transport Systems. Hannover: Gottfried Wilhelm Leibniz Universität Hannover.

Schwartz, B. (2015, December 21). *The Factors That Impact Availability, Visualized*. Retrieved from https://www.vividcortex.com/blog/the-factors-that-impact-availability-visualized

Schwartz, B., Zaitsev, P., & Tkachenko, V. (2012). *High Performance MySQL: Optimization, Backups, and Replication. 3rd Edition.* O'Reilly.

Schwemmer, R., & Neufeld, N. (2009). Implementing High Availability with COTS Components and Open-source Software. *12th International Conference On Accelerator And Large Experimental Physics Control Systems* (pp. 624-626). Kobe: JACoW.

Service Availability Forum. (2011, September 30). *Service Availability Interface.* Retrieved from http://www.saforum.org/HOA/assn16627/images/SAI-Overview-B.05.03.AL.pdf

ServTec Austria. (2015, March 19). *Rückblick 2015*. Retrieved from http://www.servtec.at/ruckblick/rueckblick-2015/

SHARE. (2013, July 9). *Don't Believe the Myth-Information About the Mainframe.* Retrieved from http://www.share.org/d/do/9005

Shenoy, A. (2015). The Pros and Cons of Erasure Coding & Replication vs. RAID in Next-Gen Storage Platforms. *Storage Developer Conference.* Retrieved from http://www.snia.org/sites/default/files/SDC15_presentations/datacenter_infra/Shenoy_The_Pros_and_Cons_of_Erasure_v3-rev.pdf

Shivakumar, S. K. (2014). *Architecting High Performing, Scalable and Available Enterprise Web Applications.* Waltham, MA: Morgan Kaufmann.

Siewert, S., & Scott, G. (2011). Next Generation Scalable and Efficient Data Protection. *Intel Developer Forum 2011.* Retrieved from https://www.researchgate.net/profile/Sam_Siewert/publication/259235933_Next_Generation_Scalable_and_Efficient_Data_Protection/links/00b7d52a8c79e7d86c000000.pdf

Siewiorek, D., & Swarz, R. (2014). *Reliable Computer Systems: Design and Evaluatuion. 2nd edition.* Digital Press.

Singh, K. (2016). *Ceph Cookbook.* Packt Publishing.

Slåtten, V., Herrmann, P., & Kraemer, F. A. (2012). Model-Driven Engineering of Reliable Fault-Tolerant Systems – A State-of-the-Art Survey. In A. Hurson, & A. Memon (Eds.), *Advances in Computers* (Vol. 91, pp. 119-204). Academic Press.

Sloan, J. (2005). *High Performance Linux Clusters with OSCAR, Rocks, OpenMosix, and MPI.* O'Reilly.

Software Advice. (2015). *Pricing Guide: Warehouse Management Systems.* Retrieved from http://www.softwareadvice.com/imglib/lightbox-download-assets/warehouse_management_pricing_guide_2015.pdf

Son, D., Chan, Y., Choi, H., Kim, W., & Higuera, A. (2015). A Study of Building a New Warehouse Control System Architecture. *International Journal of Advanced Logistics, 4*(3), 145-158.

Sonderegger, J., Blomberg, O., Milne, K., & Palislamovic, S. (2009). *JUNOS High Availability.* O'Reilly.

Sousa, K. J., & Oz, E. (2015). *Management Information Systems.* Stamford, CT: Cengage Learning.

Stapelberg, R. F. (2009). *Handbook of Reliability, Availability, Maintainability and Safety in Engineering Design.* Springer.

Storage Networking Industry Association. (2015, January). *Software Defined Storage.* Retrieved from http://www.snia.org/sites/default/files/SNIA_Software_Defined_Storage_%20White_Paper_v1.pdf

Sumimoto, S., Tezuka, H., Hori, A., Harada, H., Takahashi, T., & Tsukuba-shi, T. (1999). The design and evaluation of high performance communication using a Gigabit Ethernet. *13th international conference on Supercomputing*, (pp. 260-267).

Sun, M. H., & Blough, D. M. (2010). *Technical report: Fast, Lightweight Virtual Machine Checkpointing.* Georgia Institute of Technology.

Sungard Availability Services. (2014, November 21). *How to Sell "Disaster Recovery" to Senior Management: 5 Strategies.* Retrieved from http://www.sungardas.com/Documents/disaster-recovery-management-how-to-sell-dr-to-senior-management-MRP-WPS-084.pdf

Supply Chain Digest. (2013, April 23). *Is Cloud-Based WMS Ready for Prime Time?* Retrieved from Supply Chain News: http://www.scdigest.com/ontarget/13-04-23-2.php?cid=6969&ctype=content

Suresh, S., & Kannan, M. (2014, January-March). A Study on System Virtualization Techniques. *International Journal of Advanced Research in Computer Science & Technology, 2*(1), 134-139.

Takada, M. (2013). *Distributed Systems for Fun and Profit.* Retrieved from http://book.mixu.net/distsys/ebook.html

Tamura, Y., Sato, K., Kihara, S., & Moriai, S. (2008). Kemari: Virtual Machine Synchronization for Fault Tolerance. *USENIX Annual Technical Conference (Poster).*

Tanenbaum, A. S., & van Steen, M. (2003). *Distributed Systems: Principles and Paradigms, 2nd Edition.* Pearson Prentice Hall.

Tate, J., Kelley, R., Rossana, S., Maliska, R., Torolho, L., & Voigt, M. (2013). *IBM SAN Solution Design Best Practices for VMware vSphere ESXi.* IBM Redbooks.

techconsult GmbH. (2013). *Kritische IT-Systeme im Mittelstand.* Retrieved from http://www.softexpress.de/Media/seite_hardware/ProactiveCare/HP_Ergebnispr%C3%A4sentation_Kritische_IT_Mittelstand_Handout_2013.pdf

The Open Group. (2011). *TOGAF Version 9.1.* Van Haren Publishing.

Toeroe, M., & Tam, F. (2012). *Service Availability: Principles and Practices.* Wiley.

Trinitis, C., & Walter, M. (2003). Balanced High Availability in Layered Distributed Computing Systems. *14th International Workshop on Database and Expert Systems Applications* (pp. 713-717). IEEE.

van Vugt, S. (2014). *Pro Linux High Availability Clustering.*

Vision Solutions. (2015, February 17). *2015 State of Resilience Report.* Retrieved from http://www.visionsolutions.com/docs/default-source/default-document-library/2015-state-of-resilience-report.pdf?sfvrsn=0

Vision Solutions. (2016, January 26). *2016 State of Resilience Report.* Retrieved from http://www.visionsolutions.com/docs/default-source/white-papers/2016-State-of-Resilience-Report.pdf

VMware, Inc. (2016). *VMware vSphere 6 Fault Tolerance: Architecture and Performance.* Retrieved from http://www.vmware.com/files/pdf/techpaper/VMware-vSphere6-FT-arch-perf.pdf

von Neumann, J. (1956). Probabilistic Logics and the Dynthesis of Reliable Organisms from Unreliable Components. (C. E. Shannon, & J. McCarthy, Eds.) *Automata Studies, 24*, 43-98.

Wang, Y., & Pettit, S. (2016). E-logistics: An Introduction. In *E-Logistics: Managing Your Digital Supply Chains for Competitive Advantage* (pp. 3-31). Kogan Page.

Wolfenstein, K. (2015, February 20). *Industry 4.0 – The changing face of transport logistics.* Retrieved from Intralogistics: http://intralogistics.tips/industry-4-0-changing-face-transport-logistics/

www.FreeRaidRecovery.com. (2011). *Top 10 RAID Tips.* Retrieved from http://www.raidtips.com/raid-tips.pdf

Xin, Q., Schwarz, T., & Miller, E. (2005). Disk Infant Mortality in Large Storage Systems. *13th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (pp. 125-134). IEEE.

Xu, C.-Z. (2005). *Scalable and Secure Internet Services and Architecture.* Chapman and Hall/CRC.

Yang, J., & Sun, F.-B. (1999). A Comprehensive Review Of Hard-disk Drive Reliability. *IEEE Annual Reliability and Maintainability Symposium*, (pp. 403-409).

Zhu, J., Dong, W., Jiang, Z., Shi, X., Xiao, Z., & Li, X. (2010). Improving the Performance of Hypervisor-Based Fault Tolerance. *2010 IEEE International Symposium on Parallel & Distributed Processing* (pp. 10-20). Piscataway, NJ: IEEE.

Zhu, W.-D., Allenbach, G., Battaglia, R., Boudreaux, J., Harnick-Shapiro, D., Kim, H., . . . Willingham, M. (2009). *IBM High Availability Solution for IBM FileNet P8 Systems.* IBM Redbooks.

Zhu, X., Song, B., Ni, Y., & Ren, Y. (2016). Software Defined Anything: From Software-Defined Hardware to Software Defined Anything. In *Business Trends in the Digital Era* (pp. 83-103). Springer.