SACJ 36(2) December 2024
Research Article

Formalising solutions to network availability issues in low-resource environments: An offline storage design pattern for software systems

Abayomi Agbeyangi[a, b], Hussein Suleman[a]

[a] Department of Computer Science, University of Cape Town, South Africa
[b] Department of Computer Sciences, Chrisland University, Nigeria

ABSTRACT
In most software systems operating within low-resource environments, the persistently encountered challenges related to network availability often result in compromised performance or even complete system failure. To address this issue, software developers frequently resort to ad hoc measures to mitigate these challenges. To offer a more comprehensive solution, this paper introduces an all-encompassing approach in the form of a design pattern. This design pattern uses offline functionalities with localised storage, emphasising the distinction between offline storage and synchronisation, presenting a versatile and high-level methodology for overcoming the recurrent network-related challenges intrinsic to low-resource environments. We describe the use of the design pattern in some real-world use cases, thereby illustrating how a single design pattern is the foundation for many seemingly disparate software engineering approaches. This design pattern holds the potential to significantly redefine the manner in which network availability challenges are approached and addressed.

Keywords design pattern, software systems, low-resource environment, data synchronisation, delay tolerant networks

Categories • **Software and its engineering** ∼ **Design patterns, Reusability**

Email
Abayomi Agbeyangi – abayomi.agbeyangi@uct.ac.za (CORRESPONDING)
Hussein Suleman – hussein@cs.uct.ac.za

Article history
Received: 11 September 2023
Accepted: 2 August 2024
Online: 11 December 2024

1 INTRODUCTION

In modern software systems, many measures are taken to ensure that the programs and related configuration files, documentation, etc., that operate together function in a way that reduces network traffic, even on high-bandwidth networks. One goal is to ensure the effective and efficient use of the programs in the event of network downtime, particularly in low-resource

Agbeyangi, A. and Suleman, H. (2024). Formalising solutions to network availability issues in low-resource environments: An offline storage design pattern for software systems. *South African Computer Journal* 36(2), 1–30. https://doi.org/10.18489/sacj.v36i2.19976
Copyright © the author(s); published under a Creative Commons NonCommercial 4.0 License
SACJ is a publication of *SAICSIT*.
ISSN 1015-7999 (print) ISSN 2313-7835 (online)

environments where network access has been observed to be minimal (Abou-Khalil et al., 2021) and is occasionally unavailable.

A low-resource environment is one where resources are scarce, typically in low- and middle-income countries (Akhigbe et al., 2022; Kutoane et al., 2021). Often identified in low-resource environments are a lack of infrastructure, low income, affordability, a lack of basic literacy and digital literacy, and many other factors that contribute to limited access to networks and, as a result, degraded access to software systems. The speed of data retrieval is a crucial factor in user experience and a requirement in almost all commercial applications. Today, a number of factors affect response time, such as network infrastructure, protocols, hardware, software, and Internet speed. The strategic objectives of any organisation are seriously harmed when response times are impacted. To improve interoperability and efficiency in software systems owing to a purposeful or unintentional lack of Internet connectivity, an optimal way to ensure software systems function unfettered in either a resource-rich or low-resource environment is required. In the past (Nungu & Pehrson, 2011; Surana et al., 2008) as well as more recently (Phokeer et al., 2020; Suleman, 2021), attempts made to provide relief for this problem have produced some results; however, the solutions are ad hoc and there is no general framework to guide software developers in low-resource environments. The design pattern proposed in this study is centered on leveraging offline storage to provide a resilient solution that ensures continued functionality even when network access is restricted or unavailable. The pattern not only addresses the immediate challenges of low-resource environments but also offers a framework that enhances the overall robustness and efficiency of software systems across various scenarios. By distinguishing between offline storage and data synchronisation, our approach emphasises the importance of maintaining seamless operations without relying on real-time network connectivity, thus offering a novel perspective in the domain of software engineering for low-resource contexts.

The methodology for deriving this design pattern is grounded in both extensive literature review and practical experience (Phiri et al., 2012; Suleman, 2019, 2022; von Holy et al., 2017) gained from developing software systems in low-resource environments. We thoroughly examined existing methods, pinpointing areas for enhancement, which guided the development of the offline design pattern. This dual approach of leveraging theoretical knowledge and practical experience not only validates the robustness of our pattern but also ensures its applicability in a wide range of low-resource scenarios, thereby contributing a valuable framework to the software engineering community. The increasing prevalence of users' inability to maximally use software systems in low-and middle-income countries, such as remote and rural areas, disaster-stricken regions, and resource-constrained IoT devices, further strengthens our desire to have a framework as a design pattern to address the challenges when network connectivity is unreliable or limited. These challenges are particularly prevalent in some African regions, where the design pattern offers a robust solution to enhance system efficiency and reliability.

The main contribution of our paper lies in *formulating and presenting a novel offline design pattern optimised for software systems operating within resource-constrained environments*. Other

objectives of this study include:

- Addressing the critical gap in data syncing systems by providing a practical solution to enhance application performance in challenging scenarios.
- Tackling emerging challenges in future use cases, such as remote healthcare, and smart agriculture, for better and more efficient data localization through offline storage.

The pattern prioritises offline storage to sustain software functionality in low-resource environments with limited or intermittent network availability. By emphasising offline storage, our approach ensures seamless operation without real-time data synchronisation dependency. While data synchronisation is included, the core emphasis remains on offline storage as the fundamental solution for low-resource environments. The remaining sections of the paper are organised as follows: Section 2 discusses the literature review, emphasising some background concepts of offline operation, design patterns, and related approaches. Section 3 introduces the methodology, and the description of the pattern is presented in Section 4. Section 5 discusses the evaluation of the pattern through the case studies. The results are discussed in Section 6, while Section 7 concludes the paper with some future directions.

2 LITERATURE REVIEW

2.1 Offline Operation and local storage mechanism

The concept of local storage and offline operation has been used in various areas of the computer/networking sectors for a while and there are various implementation strategies depending on the use case. According to Xu et al. (2022), several people found offline operation or a hybrid mix to be a more practical alternative to relying solely on cloud-based systems. This is so because local storage offers comprehensive, security-oriented control over the data and better data access performance than cloud storage. For keeping the data that is regularly accessed, local storage is compact, quick, albeit sometimes costly to implement (Xu et al., 2022). Due to network latency and data constraints in low-resource environments, it is feasible to think of offline operation and local storage as a more general approach to software systems.

Local storage is any kind of computer storage that can be used without the need for network access. Examples of local storage devices include hard drives, memory and removable disks on end-user devices. When using a web application, local storage technology can be used to store user status, cached data, temporary data, permanent data, and more (Harjono et al., 2010).

The user experience is increasingly being prioritised by many Web applications, which are inextricably linked to offline operation and local storage solutions. Local storage is mostly used for server-side reasons, with the goal of storing locally accessible versions of Web applications that were previously kept on the server. Liu (2014) emphasises the benefits of offline operations and local storage as a solution that relieves server-side stress and can also save a significant amount of network traffic, enabling faster software data queries. With offline operation and local storage, users of software systems (mobile or desktop applications) can engage

with them even when there is no network connection because of offline synchronisation, which is facilitated by local storage. A local data store is where changes are kept. These changes are synchronised with the remote server after the device has been brought back online.

Software applications that employ online and offline synchronisation using local storage make the software useful even when the network is unavailable. On the other hand, users must wait for data to load from the servers every time they request it when using an online-only, no-local-storage approach. This is an unpleasant experience because mobile or broadband networks in many low- and middle-income countries are generally slow, unreliable, and occasionally unavailable. Even when the data is accessible, loading it takes a few seconds. This contradicts the objective of providing a speedy and dependable software experience to software system users even in low-resource environments.

Despite the fact that the majority of people have access to the Internet in most advanced countries, there are still situations where software systems might not be able to connect to the Internet. The basement of an industrial facility or inside a plane are two notable examples, and every organisation must plan for contingencies. To address this issue, some modern applications, notably mobile applications, have introduced the concept of offline-first architecture. At any time the device is offline, the app remains semi-usable according to its architecture; for example, users of a messaging app can see their list of chats and message history. However, fresh messages cannot be sent.

The following are some of the foundational ideas of the offline-first architecture:

- Only data from local storage (e.g., database and key-value store) is displayed in the application's user interface.
- The application user interface never asks remote devices for data directly – all data access is via local storage.
- A synchronisation action keeps the local data current with the remote device data.

In most typical local storage synchronisation operations during the availability of networks, an application would check the remote device for updated or new data before storing it locally. Synchronisation operations can be started in a variety of ways, such as when a user opens a screen or receives a push notification saying that fresh data is available on the remote device. In contrast, some of the existing data syncing systems have been seen to focus on high-resource environments mostly (Jannes et al., 2021; Singh & Hasan, 2019). We observed a lack of a dedicated approach targeting low-resource environments, making our approach unique and valuable.

In order to achieve long-term gains in data quality and software system performance in low-resource environments, Ramanujapuram and Malemarpuram (2020) argued that the absence of an essential resource, such as a good network, is a significant barrier. As can be seen in low-resource environments, particularly in rural African regions, it is believed that despite the fact that many Internet users now have access to high bandwidth and high-speed networks,

there are still some low-resource environments where broadband services are either incredibly expensive or simply unavailable (Jebessa & Alemayehu, 2009; Tang et al., 2021; USDA, 2019).

2.2 Design patterns

Design patterns are blueprints that provide solutions to specific problems encountered during software development (Hussain et al., 2018; Leimeister et al., 2021; Meheden et al., 2021). Leimeister et al. (2021) believe that design patterns support developers in addressing technological challenges and crafting solutions by codifying best practices and making them applicable for future use. This plays a critical role in both practice and research. They represent a fundamental aspect of modern software engineering, offering structured solutions to recurring design challenges that span various domains (Hussain et al., 2018; Jiang & Mu, 2011; Seidl et al., 2017). Their purpose extends beyond individual projects (Leimeister et al., 2021), as they address recurring issues across diverse applications. While it is true that prior solutions might exist for certain problems, design patterns offer a standardised framework that transcends isolated solutions (Barakat, 2019). They encompass the collective experience and insights of the software community, distilled into templates that facilitate informed decision-making and expedite the development process.

The study by Gamma et al. (1995), mostly called Gang of Four (GoF), introduced the concept of software design patterns. In 1995, they released a book that served as a framework for discussing design patterns. These design principles are commonly utilised in the creation and operation of large software systems (Rasool & Akhtar, 2019). The GoF design patterns have become an industry-standard reference and offer a set of tried-and-tested answers to common design issues in object-oriented software development (Rath et al., 2019).

The rigour of a design pattern approach lies in its holistic consideration of factors that extend beyond the immediate problem, including scalability, maintainability, and adaptability. Therefore, even when solutions have been suggested in prior work, design patterns offer a refined lens through which to examine, evaluate, and potentially enhance those solutions. This paper's contribution, framed within a design pattern context, provides a systematic, tested perspective that augments existing solutions and serves as a robust foundation for addressing the complex challenges software systems face in low-resource environments, specifically under limited or no network access. The design pattern approach, which is the basis of this paper, is not intended to provide new experimental validation of a specific solution. Instead, it is to extrapolate from multiple past experimental results to specify a generalizable approach that can apply to future problems.

2.3 Related Approaches

In the literature, efforts have been made, though not explicitly as design patterns, to address issues with network connectivity (Nungu & Pehrson, 2011; Phokeer et al., 2020; Suleman, 2021; Surana et al., 2008), offline operation (Demers et al., 1994; Kistler & Satyanarayanan, 1992;

Liu, 2014; Satyanarayanan, 1989; Satyanarayanan et al., 1990; Terry et al., 1995), and software resilience (Cámara et al., 2014; Quinn et al., 2015) in low-resource environments. Notable among these techniques for offline operations is *Coda* (Satyanarayanan, 1989), a distributed file system design that helps improve software performance by ensuring data availability even during server crashes or network partitions. According to Satyanarayanan et al. (1990), it was designed for large-scale distributed computing environments, ensuring resiliency to server and network failures through server replication and disconnected operation mechanisms. One other notable historical technique for efficiently transferring and synchronising files between two locations is *RSync* (Tridgell & Mackerras, 1996). It is particularly useful for offline operation, as it allows users to synchronise files across devices or networks, even when the network connection is unreliable or intermittent. It achieves this by only transferring the parts of files that have changed since the last synchronisation, reducing the amount of data transferred, and minimising network overhead. Other offline data syncing frameworks that use permanent integration and continuous synchronisation to local storage include *Azure Offline Sync*[1], *Realm Mobile Database*[2], *Back4App*[3], *AWS Amplify*[4], *Firebase*[5], and *Kinvey*[6]. The study of historical tools and techniques, as well as common offline data syncing frameworks, has provided valuable knowledge about the methods and approaches used in developing software systems that can operate with limited network access. These tools and techniques represent significant milestones in the development of offline functionality in software systems.

The request/reply pattern (Brahma & Sadhya, 2022; Buschmann et al., 2007) for software system communication is a similar approach, in which one component, the client, submits a request to another, the server, who then answers with a reply containing the required data or by carrying out the requested action. It is frequently employed in client-server architectures, distributed systems, and service-oriented architectures (SOA) (Abdellatif et al., 2018). The idea is also similar to what can be achieved in an environment of edge computing, where base station edge servers are placed close together (Zhang et al., 2020). Due to this, high-speed links can be used for data sharing and communication between neighbouring edge servers.

The mobile offline-first pattern (Biørn-Hansen et al., 2018; Vanhala, 2017) is another important pattern that has attracted attention as a valuable method in mobile app development. It emphasises the importance of developing applications that can perform well in offline environments, allowing users to interact with the app even when connectivity is intermittent or non-existent. Other approaches and perspectives include device-to-device file transfers (Scott, 2016; Z. Wang, 2022) and service-based mobile application frameworks (Brunette et al., 2017), which encompass an event-driven offline approach. These approaches allow for direct communication between devices, enabling efficient file transfers without needing an Internet con-

[1] https://learn.microsoft.com/en-us/azure/developer/mobile-apps/azure-mobile-apps/howto/data-sync
[2] https://www.mongodb.com/docs/realm-legacy/products/realm-database.html
[3] https://www.back4app.com/
[4] https://aws.amazon.com/amplify/
[5] https://firebase.google.com/
[6] https://devcenter.kinvey.com/rest/guides/core-overview

nection. On the other hand, service-based mobile application frameworks utilise a centralised server that handles offline events and synchronises data across devices once connectivity is restored. These approaches provide flexibility and robustness for offline mobile applications. However, our proposed offline design pattern extends beyond the mobile environment to provide a generalised solution that can be used for a wide range of software systems that face intermittent network access and resource constraints.

Examining the four elements of the proposed design pattern and their connection to relevant methods like *Coda*, *RSync*, and selected Offline Data Syncing Frameworks (*Azure Offline Sync*, *Realm Mobile Database*, *Back4App*, *AWS Amplify*, *Firebase*, and *Kinvey*), these tools ensure the availability of data by synchronising crucial data locally during software system installation. The 'pre-populate' process establishes the groundwork for offline functionality, allowing users to retrieve and modify data saved locally without depending on network connectivity. During the 'offline operation' phase, these tools enable apps to operate smoothly in situations where there is no Internet connection, allowing users to access and modify data stored locally. This can be seen with *Firebase* operations that allow applications to access and modify data stored locally, ensuring uninterrupted functionality without network connectivity. The 'update' stage enhances the continuous functionality by synchronising data changes with remote servers when the connection is restored. During the 'post-populate' stage, these tools ensure data consistency across local and remote sources by synchronising any locally made changes with the server-stored information. This indicates the offline operation cycle within the offline design pattern. Furthermore, caching, MQTT (Message Queuing Telemetry Transport) (Haque et al., 2021) and CoAP (Constrained Application Protocol) (Herrero, 2020) are useful technologies for enhancing efficiency and facilitating communication in online contexts. Caching mitigates high traffic rates by actively storing frequently accessed content in users' local memories (Maddah-Ali & Niesen, 2014; J. Wang, 1999). MQTT[7] is an OASIS standard messaging protocol designed for the Internet of Things (IoT). It is a lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with limited code size and bandwidth constraints (Hunkeler et al., 2008). CoAP (Bormann et al., 2012), unlike MQTT, is another lightweight messaging protocol specifically designed for constrained environments, with a focus on low power consumption and simplicity. Both protocols offer efficient communication solutions for IoT devices, but CoAP is more suited for resource-constrained environments where minimal code size and energy consumption are critical factors (Thangavel et al., 2014). However, they are strongly dependent on Internet connectivity for the synchronisation of data. The offline design pattern we offer gives priority to local storage and operation, ensuring smooth functionality even in situations when Internet connectivity is limited or unavailable. Our pattern prioritises offline storage, providing a dependable standardised solution in the form of a design pattern for software systems that operate in low-resource environments. It overcomes the constraints of internet-dependent technologies and improves system resilience in offline situations.

The studies that focus on technology deployment in low-resource environments provide

[7] https://mqtt.org/

invaluable insights into the strategies, challenges, and best practices associated with software system effectiveness. Pentland et al. (2004) conducted a notable study on DakNet, highlighting creative methods to address connectivity challenges in rural regions. The study emphasises the effectiveness of community-based networks (such as the Inethi project (Phokeer et al., 2020)) in easing the distribution of data. Similarly, the deployment of the Open Data Kit (ODK), as highlighted in Hartung et al. (2010), Brunette et al. (2013), and Brunette et al. (2017), provides essential frameworks for designing information services tailored to the needs of developing regions. Likewise, research conducted by Chandwani and Kumar (2018) provides insight into the significance of telemedicine in areas with limited resources, highlighting the necessity of establishing interconnected systems to enhance healthcare provision. Insights from these studies enrich our understanding of the challenges and opportunities inherent in technology deployment in low-resource contexts.

While influenced by the mobile offline-first concept and other related approaches, our design pattern broadens the basic concepts of these approaches to accommodate a broader range of scenarios applicable to various software engineering domains. We have improved and expanded the fundamental concepts of offline functionality and local storage usage, making them applicable to a wide range of scenarios and settings. The inclusion of this broader viewpoint not only strengthens our design pattern's conceptual framework but also improves its versatility and adaptability in addressing the intricate issues faced in low-resource environments and beyond. The design pattern provides a standardised solution to the persistent issue of network connectivity limitations. As such, the design pattern represents a significant standardised solution to the challenges of network connectivity constraints.

3 METHODS

The design pattern is developed through extensive experience and concepts from related approaches in the literature, addressing the specific challenges of creating software systems in low-resource environments. The approach is based on the qualitative synthesis of practical insights and existing theoretical frameworks. The design pattern seeks to offer a resilient solution for maintaining software system functionality in scenarios of intermittent or unavailable network connectivity, emphasising the importance of offline storage over real-time synchronisation.

The proposed pattern is evaluated through a series of six case studies, which illustrate the practical application and effectiveness of the pattern in real-world scenarios. These case studies were chosen based on specific criteria to ensure they accurately depict the challenges faced in low-resource environments. The selection criteria for the case studies include:

i. Relevance to low-resource environments – demonstrate scenarios where network connectivity is limited or intermittent

ii. Diversity of application domains – chosen from different domains, such as data collection, navigation, and content distribution

iii. Implementation of offline storage – showcase the implementation of offline storage as a critical component

In evaluating the pattern a uniform methodology was employed across all chosen case studies. This entailed thoroughly examining how each case study integrated the design pattern and its sub-patterns, emphasising the four sub-patterns: pre-populate, offline operation, update, and post-populate. For each case study, we examine:

- The specific challenges addressed by the pattern.
- The implementation process and any adaptations made to fit the unique requirements of the case.
- The outcomes and effectiveness of the pattern in enhancing system functionality and resilience

By analysing these aspects, we can identify commonalities and differences in applying the pattern, comprehensively evaluating its utility and versatility. This approach validates the proposed design pattern and offers insights into its practical implications and potential for further refinement based on future use in various low-resource environments.

4 THE OFFLINE PATTERN AND SUB-PATTERNS

To begin the process of designing an offline system, the first step is to manage the data that is processed at the remote end and synchronise it to the local storage. Processing a particular collection of data along with a time frame, location, and other parameters in order to accomplish efficient data synchronisation can lead to optimisation of the data. Since it determines the application's overall functionality, the synchronisation approach is a key component of any offline infrastructure. The synchronisation approach is what determines how much weight offline data should be given compared to online data. The network connectivity options available to the user, the importance of the data, and any other requirements for continuity should all be taken into consideration when selecting the appropriate synchronisation approach. Figure 1 illustrates the key components of the pattern, for which an overview follows:

- **Pattern Name** – Offline pattern
- **Pattern Description** – The pattern to address network availability issues of software systems in low-resource environment
- **Problem** – Inability of software systems to function maximally due to partial or complete unavailability of the external network
- **Solution** – Incorporate the use of offline operation and local storage in the architecture of software systems

- **Use When** – Shall be used when access to the network is poor or unavailable
- **Example** – Examples are illustrated as case studies (Section 5).

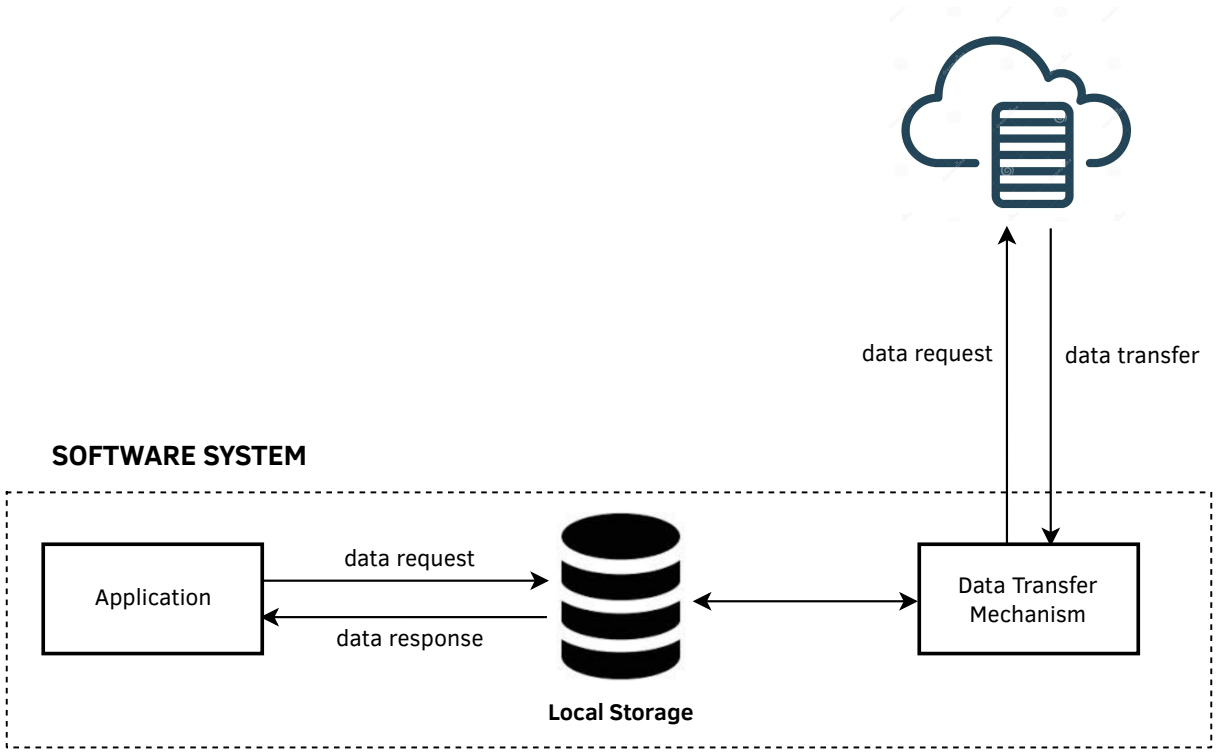


Figure 1: The Design Pattern Components

4.1 Offline storage

When users of software systems try to access stored data and services online in low-resource environments, there seem to be times when Internet access is limited or unavailable, thus impairing the ability of users to use the software system. With the advantages presented by the use of local storage, users of software systems now have the chance to deploy, evaluate, and use new software and application tools offline even when they have little to no access to the Internet.

> *How can software systems optimised for low-resource environments take advantage of on-premises data storage (local storage) to overcome the persistent challenge of intermittent network connectivity?*

Data caching (Copps et al., 2021; Zhang et al., 2020), online/offline mode and computation offloading are a few techniques used in low-resource environments to address software system

network unavailability. There is no need to download or search a database for a resource that has been cached (Durner et al., 2021). Even when a query is optimised, the database still needs to be queried, which involves network traffic and potentially expensive database-level computation (Mertz et al., 2021). Avoiding these steps reduces application latency and increases user responsiveness. Additionally, it makes it possible for the application to manage and function effectively under a growing or increased workload. The proposed approach to ensure data availability at all times is to initially store the needed data in the local storage before requesting it again. The software systems can then operate offline without the need for network access by using the data that was previously stored locally for subsequent requests for the same data. This prevents the need to re-fetch data. The application requests fresh data from the stored data when it needs to use it. If the information is present in the local storage, it returns the instance of the information that was previously saved. If the local storage does not have the data, an attempt could be made to connect to the network to access an online data repository (which could be a file, a database, etc.), fetch the data and then return it to the application. When that information is subsequently requested again, the stored data is returned. This process repeats for the duration of the application's lifecycle.

Local storage enables Web-based programmes to run offline, saving data in local storage that can be accessed without requiring a network connection. In comparison to a server-side database, local storage has a number of benefits, such as quick response times, offline functionality, and reduced network latency.

Therefore,

> *Use local storage as a primary data storage mechanism. To support offline operation, where necessary, transfer data to local storage, perform updates during operation and/or transfer data from local storage to remote devices.*

Initially, data needs to be transferred to a local storage device by some mechanism, whether it is network-based or not. Then, at any time network connectivity is available, a data update can occur, and the locally stored content can be updated if changes have occurred to the remote device's data.

4.2 Detailed Description of Sub-Patterns

In order to provide a thorough understanding of the proposed design pattern, it is further divided into four essential sub-patterns. Each sub-pattern addresses specific data management and system functionality aspects in low-resource environments with intermittent or limited network connectivity. By outlining these sub-patterns, we intend to present a clear framework that can be consistently applied to various scenarios, enhancing the resilience and efficiency of software systems operating under constrained conditions. In the following sections, we will explore each sub-pattern in detail, explaining their roles, implementation strategies, and the benefits they bring within the broader design pattern.

As illustrated in Figure 2, the sub-patterns are:

a. Pre-Populate

b. Offline Operation

c. Update

d. Post-Populate

Pre-Populate

During installation, an application must ensure that the minimum data required to operate the application can be found locally. Figure 2(a) illustrates this sub-pattern, which ensures the local storage keeps sufficient data to start the application locally. This must be done in order to guarantee that the essential data for the user interface can be found in the local storage. This ensures that the application can start up regardless of whether there is an active Internet connection.

Offline Operation

During normal operation, an application will only read data from and write data to local storage. This ensures that the application will work without a network connection or online access to remote devices. This is as described in Figure 2(b).

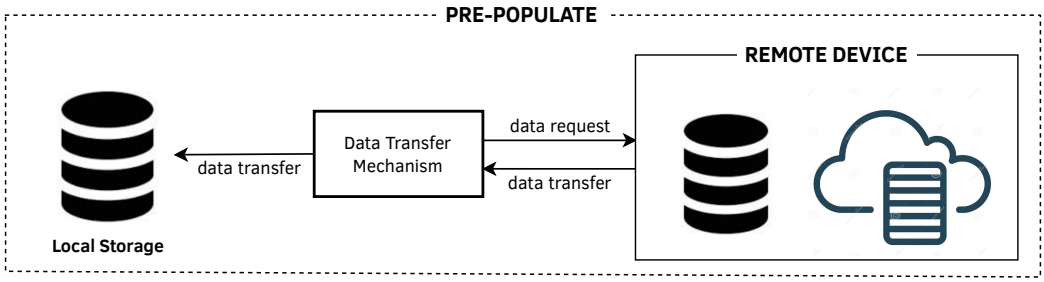
Update

This process connects to the data source stored on the remote device so that new modifications can be synchronised with the local storage. It is the task of the pull operation (data request), which is sent out, to retrieve updated data from the remote location and save it in local storage (Figure 2(c)). Data updates can also be sent from the local storage to the remote device.

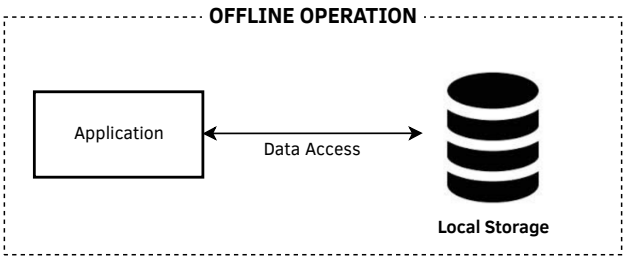
Post-Populate

It may be necessary to ensure that the contents of the local storage are consistent with the information that is stored on the remote device, particularly when data has changed locally. If the application produces data rather than consumes it, a transfer of this data may be required to some remote device. The data set in the local storage is moved using a transfer operation (Figure 2(d)).

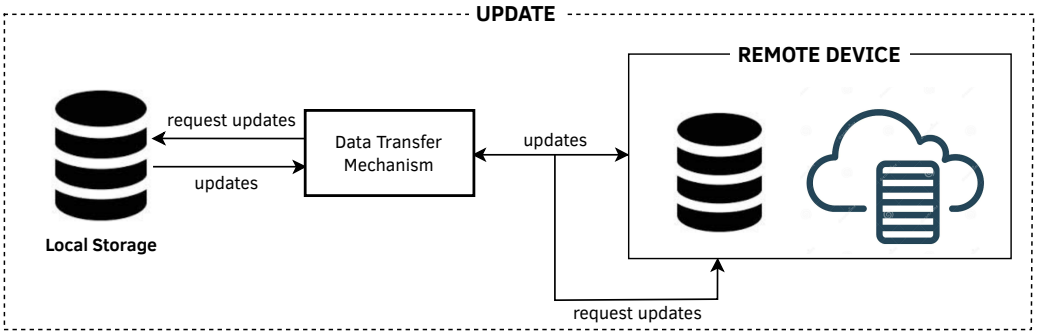
https://doi.org/10.18489/sacj.v36i2.19976



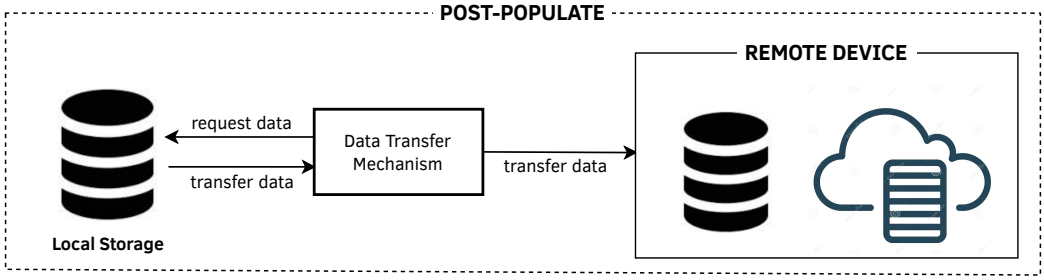
(a) Pre-Populate sub-pattern



(b) Offline Operation sub-pattern



(c) Update sub-pattern



(d) Post-Populate sub-pattern

Figure 2: Sub-Patterns of Offline Pattern Modules

https://doi.org/10.18489/sacj.v36i2.19976

The combined sequence diagram illustrating the offline pattern is shown in Figure 3.

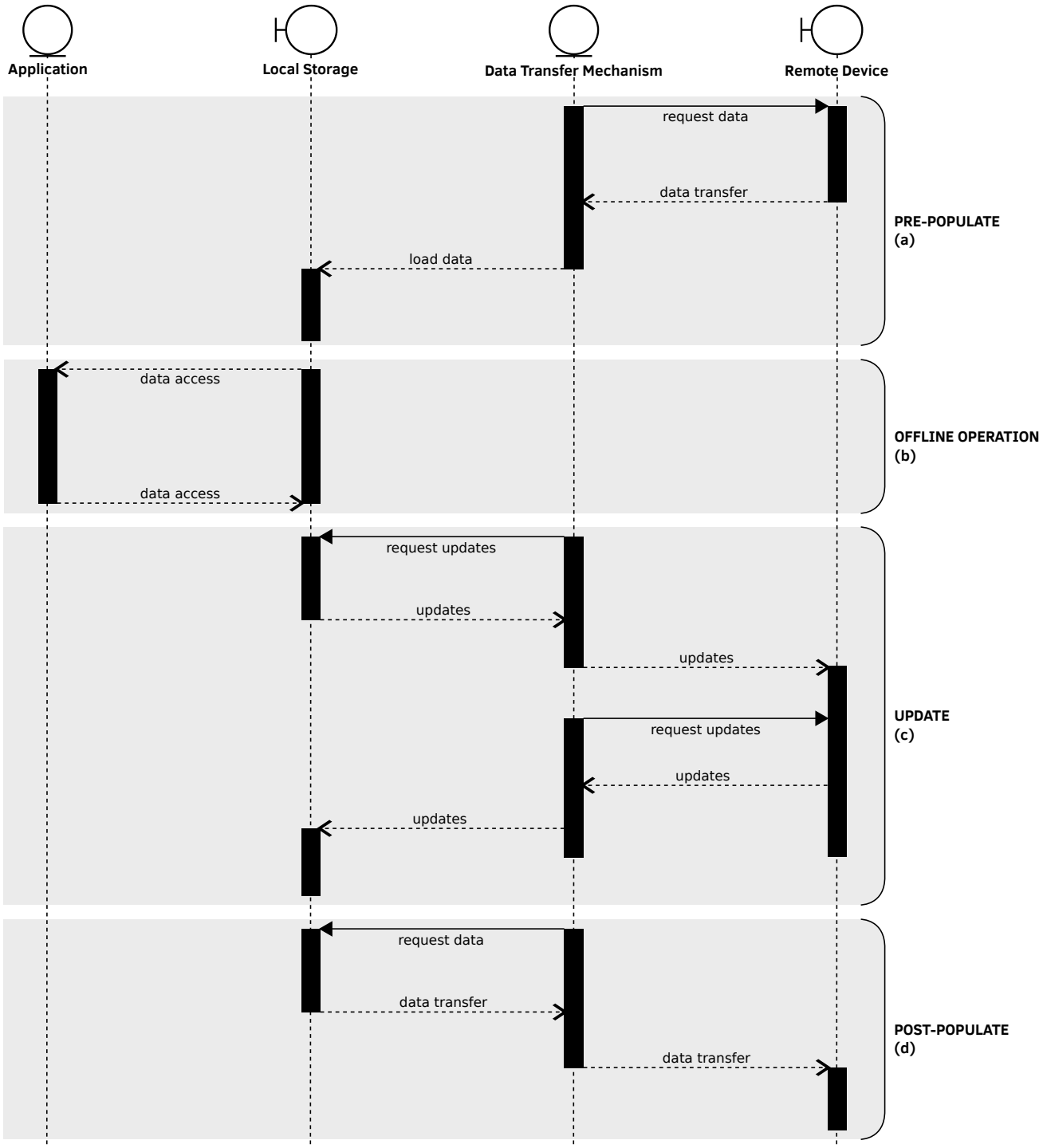


Figure 3: Offline Pattern Flow Process

https://doi.org/10.18489/sacj.v36i2.19976

4.3 Synchronisation

With all the sub-patterns, the data transfer mechanism of the application should be in charge of controlling data transfer to and from remote devices. Synchronisation methods should be used to ensure that data is synchronised in both directions between local storage and remote devices, using push and pull notifications and incorporating conflict resolution mechanisms. Data consistency conflicts can occur when the same data has been updated on local storage and on a remote device – in such a case, the specific use case should include a resolution mechanism. A use case that disallows simultaneous updates to data on local storage and remote devices will attain greater data consistency.

To avoid data consistency errors on local storage and remote devices, each request and response operation can include data and operation timestamps to enable correct sequencing of operations. Timestamps will also allow partial updates of data to be transmitted and maintained. This strategy protects against inconsistencies while making the most efficient use of available time, processing, and networking resources.

It is appropriate to consider deletions, whether they occur online or offline, to be the same as modifications. In all sub-patterns, the list of data that has been modified both on the remote device and in the local storage must be maintained, along with the date/time on which each modification was made. This is necessary in order to maintain records of modifications made. During an *update* or *post-populate* operation, recently modified data in local storage can be sent to and from a remote device on the basis of the relevant timestamps.

In some use cases, it is not uncommon for users to have more than one device, which can make managing offline applications and local storage more of a challenge. When a user attempts to synchronise data from two different devices to the remote device, a data consistency conflict may result. This conflict needs to be resolved in the *data transfer mechanism* of the remote device. This can be accomplished using a client-wins system[8], a server-wins system, a time-based resolution, or a version control system.

When using the client-wins approach, the data stored on the local storage will take precedence over any previously stored information in the remote device. In the case of the server-wins approach, the data on the remote device will take precedence over the potentially contradictory data from the local storage. The time-based resolution determines whether the most recent data will be stored in the application regardless of whether the data is currently stored in the local storage or the remote device. Version-controlled resolver functions in a manner that is analogous to that of the git repository. If a disagreement arises, the user should be given the option to select which version of the data should be maintained within the system.

5 CASE STUDIES

The case studies presented are selected based on their relevance to low-resource environments, particularly within the African context. These include projects like OAI-PMH, iNethi, GitHub

[8] https://www.bigthinkcode.com/insights/offline-first-application

Desktop, Freedom Toaster, Navigation and Mapping, and Data Collection, which illustrate the practical applications and benefits of the offline design pattern.

5.1 Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH)

The Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) is a standard protocol in the digital libraries community that is used for transferring metadata among digital repositories and metadata-based service providers (Lagoze & Van de Sompel, 2003). The protocol was developed in response to a need to aggregate metadata from multiple repositories to create high-quality discovery tools for researchers. Federation of search queries was the previously-used technique, but its reliance on the network meant that the number of sources was equal to the number of potential sites of failure, and failures were not uncommon. In contrast, harvesting requires a once-off transfer of metadata to an aggregator, which can then provide discovery services, such as search engines, to end-users without further use of the data provider repository.

The remote device is referred to as the data provider, and the local device is the service provider. OAI-PMH predominantly uses the *pre-populate* sub-pattern (Figure 2(a)) as the first request for data is often for the entire dataset. The first request has a format such as:

```
http://some.server/OAI?verb=ListRecords&metadataPrefix=oai_dc
```

Thereafter, the service provider will request regular (often daily) updates from the data provider using a protocol request that includes a last-modified timestamp each time, such as:

```
http://some.server/OAI?verb=ListRecords&metadataPrefix=oai_dc&from
    ↪ =2023-04-15
```

These operations are robust in that network and remote device failures do not affect the ability of the service provider to provide services to its users. Also, since only timestamp-driven updates are transferred, updates consume very little Internet bandwidth and duplication is minimised. Thus the protocol is robust and resource-efficient, even in low-resource environments.

Data is transferred only in one direction so there is no *post-populate* sub-pattern. Data consistency is also not a problem because all updates and deletions are tracked using timestamps, and updates only occur in one direction (from data provider to service provider).

The stability of OAI-PMH also allows for hierarchical aggregators where meta-aggregators will obtain metadata from data providers that are themselves aggregators. This approach is used in the Networked Digital Library of Theses and Dissertations, where its international Union Archive (Suleman, 2004) uses OAI-PMH to obtain metadata from universities as well as university consortia and even some national consortia (like the UK's British Library and the

Brazilian national ETD project). Each of the consortial projects maintains its own data aggregator for member sites. In such a hierarchical system, updates filter from the source through multiple layers of aggregation. Data consistency is weakest at higher levels of aggregation, but most aggregators update their local stores daily so service providers are at most 2-3 days out of sync with the source archives of electronic theses.

Most service providers provide online search services to users, but do this on the basis of local storage and offline operations (Figure 2(b)) so only the end-user interface is network-dependent and all data processing and data access is local. This substantially improves the quality of service, resource requirements and robustness for such service provision.

5.2 Navigation and mapping

Navigation and mapping software applications allow end users to view maps and navigate road networks in real-time; most such tools work in low-resource environments because of a local storage and offline operation basis of operation.

The navigation and mapping applications following the offline pattern can store and retain map data locally on the device by making use of the sub-pattern *pre-populate* (Figure 2(a)). This may include maps of entire regions or simply the places that the user is most likely to visit during their trip. If the user's map data is stored locally, then even if there is no network connection, the user can still consult the map and navigate even though there is no network connection. Additionally, the application is able to locally record route information on the device itself. This can include information such as distance, estimated journey times, and turn-by-turn directions to the destination. Because this information is stored locally, it allows the user to continue following the journey even if there is no network connection. Locally on the device, the user's preferences can be saved based on the past log entries of searches. This can include things like recommended routes, favourite destinations, and settings like voice guidance and distance units, among other things. Because these preferences are stored locally, the user does not have to re-enter this information each time they use the application. In addition to this, the users will have access on their device to a number of interesting and significant points of interest, which can continue to be updated at any time the application can access the Internet for new updates (using the *update* sub-pattern: Figure 2(c)) This can include establishments such as hotels, restaurants, and service facilities for gasoline.

5.3 iNethi Community Network

The iNethi Community Network is a project in a peri-urban township of Cape Town, South Africa, that is community-driven and attempts to deliver affordable and accessible Internet connectivity to places that are not currently served (Phokeer et al., 2020). The primary purpose of the iNethi Community Network's local storage component is to enable offline content caching and to provide local service provision.

The iNethi Community Network can use the offline pattern for content caching to locally cache frequently visited material (*pre-populate* sub-pattern: Figure 2(a). This can include things that are often accessed by community members, such as web pages and multimedia files. The system first determines if a requested piece of content is present in local storage when a user requests it. If so, accessing the Internet and relying on the network connection is not necessary because the content can be obtained straight from the local storage.

The offline pattern can be used to optimise the use of resources in the iNethi Community Network. For instance, during periods of high network connectivity, commonly used data or resources can be preloaded onto the local storage, making them immediately accessible for offline access later on when the network connection is weak or unavailable (*update* sub-pattern: Figure 2(c)) This helps to decrease the need for frequent queries to external services, saves bandwidth, and enhances the system's overall performance and responsiveness.

The offline pattern can be used for offline data collection by the iNethi Community Network as well. Members of the community, for instance, can gather data using applications or tools that are installed locally and then save it in local storage. The data can be synchronised with the central server or online database for additional processing or analysis when a network connection becomes available (Figure 4).

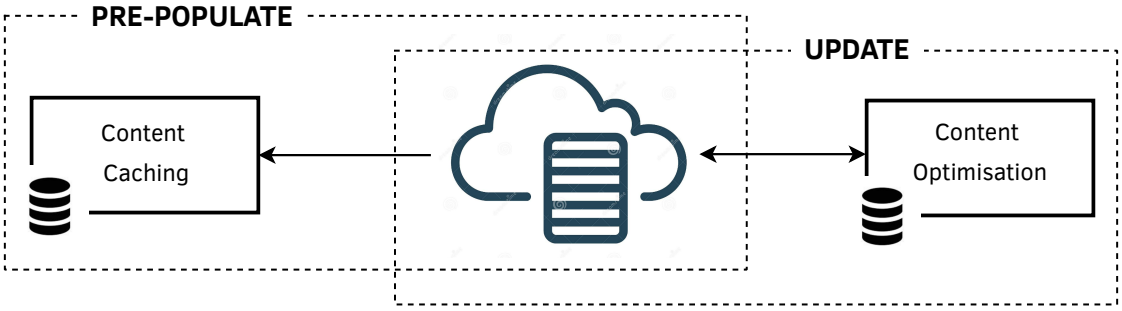


Figure 4: Use-case for iNethi Community Network

5.4 GitHub Desktop

For managing Git repositories on a local computer, a programmer can use a Git client such as GitHub Desktop[9]. For offline use and to support managing Git repositories, it has features that allow data to be stored locally on the user's computer.

In the GitHub desktop application, a user can still view and use a repository offline if they have previously used GitHub Desktop to clone it, and the repository files are stored locally on the local machine (*pre-populate*: Figure 2(a)). Thus, even without an online connection, they may use GitHub Desktop to monitor the commit history, edit files, create and manage branches, and carry out other Git-related tasks inside the local repository.

[9] https://github.com/

Furthermore, since all branch-related information and commit history are stored locally, users can continue to create, switch between, and merge local branches using GitHub Desktop even when there is no network connection. However, an Internet connection would be necessary in order to push changes to or get updates from a remote repository, which can use *update* sub-pattern (Figure 2(c)). Figure 5 further illustrates this.

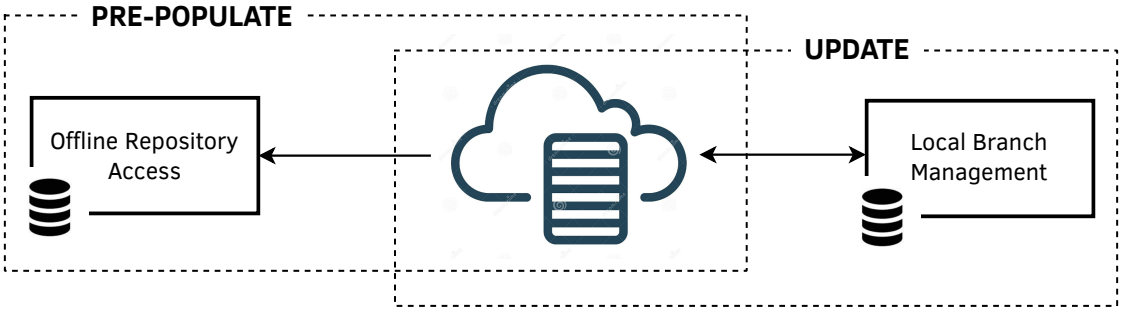


Figure 5: Use-case for GitHub desktop

5.5 Freedom Toaster

A free and open-source project called the Freedom Toaster[10] enables users to quickly replicate digital data, like software, documents, and multimedia files, onto their own blank CDs or DVDs. Its goal is to facilitate access to knowledge and information in locations where Internet access may be difficult to find or expensive.

The operation of the Freedom Toaster is based on a large collection of CD or DVD images being stored on its hard drive (local storage) for distribution to users who physically visit the Freedom Toaster. A visitor inserts a blank CD or DVD into a DVD writer on the device and uses the touch-screen or other user interface to burn or "toast" a copy of free software or free data, such as educational material.

The Freedom Toaster conceptually uses the offline pattern twice. In the first instance, the Toaster itself uses the *pre-populate* sub-pattern (Figure 2(a)) to maintain its local collection of CD/DVD images. Then, each visitor also uses the same offline pattern in burning their CDs/DVDs as they leave with portable data stores that are pre-populated and can support offline access on their home computers. There is no post-populate or update (Figure 6).

[10] https://brandsouthafrica.com/108893/freedom-toaster/

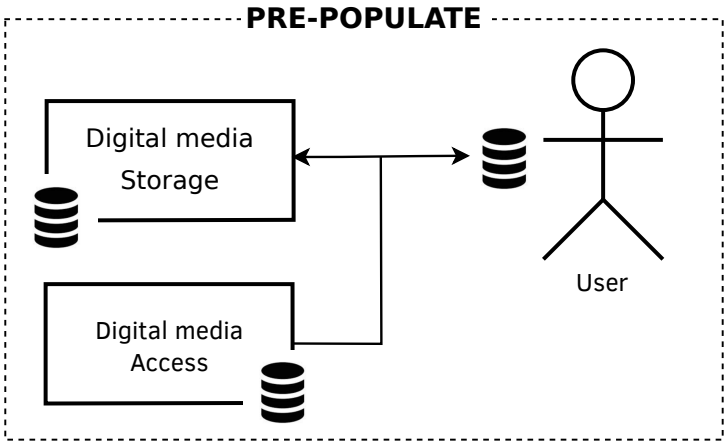


Figure 6: Use-case for Freedom Toaster

5.6 Data Collection

In the last use case, a need may arise where data is collected locally and transferred to a remote device. This scenario uses the *post-populate* sub-pattern. The data is collected locally for onward transfer to a remote device.

The *post-populate* pattern (Figure 2(d)) is used for data collection tasks even in the absence of network connectivity by collecting data locally on the local device during network unavailability and synchronising with the remote device once the network becomes available. This enables data gathering while network connectivity fails and, afterwards, synchronisation with the remote device (Figure 7).

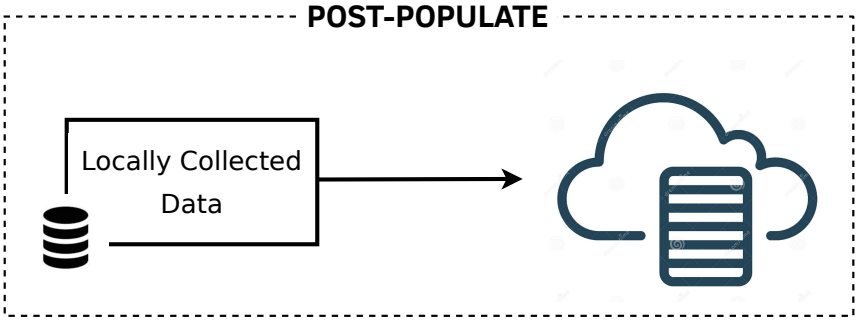


Figure 7: Use-case for Data collection

Bird et al. (2014) developed the Aikuma mobile application that runs on smartphones to collect and annotate audio data, as an example of the data collection use case in low-resource environments. Their application was deployed in remote villages with no external network connectivity, so the application ran completely offline using local storage. To avoid data

loss, data was exchanged within a small team of people, but this data could only be shared externally at a later point in time.

5.7 Comparative Analysis of Sub-Patterns

In analysing the application of the offline pattern across various case studies, it becomes evident that the effectiveness and necessity of each sub-pattern—pre-populate, offline operation, update, and post-populate—vary depending on the specific requirements and constraints of each case. For instance, in the OAI-PMH case study, the pre-population sub-pattern played a pivotal role in ensuring that essential metadata was available offline, thereby enabling uninterrupted access to digital archives. Conversely, the iNethi project prioritised the offline operation sub-pattern to empower community members to access and share information without depending on continuous Internet connectivity.

Similarly, the GitHub Desktop case demonstrated the importance of the Update sub-pattern, as it required regular synchronisation of local repositories with remote servers to maintain code consistency. On the other hand, the Freedom Toaster project highlighted the efficiency of the Post-Populate sub-pattern, guaranteeing the updating and alignment of locally stored software packages with external repositories when reconnecting to the Internet. The navigation and mapping use case showcased the comprehensive application of all sub-patterns, addressing the crucial requirement for preloaded maps, offline functionality, and periodic updates. Meanwhile, the data collection use case highlighted the importance of both the offline operation and update sub-patterns, with data collection occurring offline and subsequent synchronisation with central databases. Table 1 highlights which sub-patterns were used and their significance in each context.

Table 1: Comparative analysis of sub-patterns in each case

Case Study/Use Case	Pre-Populate	Offline Operation	Update	Post-Populate
OAI-PMH	High	Medium	Low	Low
iNethi	Medium	High	Low	Medium
GitHub Desktop	Medium	High	High	Medium
Freedom Toaster	High	High	Medium	High
Navigation and Mapping	High	High	High	High
Data Collection	Medium	High	High	Medium

As shown in Table 1, the Pre-Populate and Offline Operation sub-patterns played a crucial role in ensuring initial data availability and enabling offline functionality in the OAI-PMH and Freedom Toaster projects. Both the OAI-PMH project, which required less frequent updates, and the Freedom Toaster project, which needed consistent updates for software version maintenance, relied heavily on these sub-patterns. Similarly, the iNethi and GitHub Desktop projects prioritised offline operations to facilitate continuous usage without connectivity, with

GitHub Desktop also emphasising the Update sub-pattern to synchronise local and remote repositories. On the other hand, a comprehensive application of all sub-patterns was necessary for the navigation, mapping, and data collection use cases. While navigation and mapping relied on constant updates and synchronisation to maintain accurate and relevant map data, data collection focused on offline operations for uninterrupted data gathering and subsequent synchronisation to ensure data integrity. These case studies highlight the flexibility and adaptability of the offline pattern in addressing diverse challenges in low-resource environments. This analysis not only demonstrates the practical use of the pattern but also offers a framework for its application in future projects.

In each case study, the decision to include or exclude specific sub-patterns was based on the unique requirements and operational settings of the projects (see Table 2). In the OAI-PMH case, the *Pre-Populate* sub-pattern was given priority due to the need for offline availability of metadata for archiving purposes. The Update and Post-Populate sub-patterns were considered less crucial due to the static nature of the archived data. For the iNethi project, *Offline Operation* was of utmost importance, as the project aimed to deliver Internet-like services in areas with unreliable connectivity, making continuous offline functionality essential.

Table 2: Justification for sub-pattern use/omission across case studies (orange coloured patterns were omitted, the rest were used).

	Pre-Populate	**Offline Operation**	**Update**	**Post-Populate**
Case study: **OAI-PMH**				
	Ensured metadata is available offline for archival purposes.	Enabled interaction with data without Internet connectivity.	Infrequent updates needed for archived metadata.	Minimal need for synchronisation due to the static nature of metadata.
Case study: **iNethi**				
	Initial data availability is crucial to provide Internet-like services.	Allowed continuous service delivery in areas with unreliable connectivity.	Static nature of content meant updates were less critical.	Ensured data consistency upon reconnection.
Case study: **GitHub Desktop**				
	Facilitated initial setup with necessary data for development workflows.	Critical for uninterrupted development workflows without connectivity.	Frequent synchronisation of local and remote repositories is necessary.	Maintained consistency between local changes and remote repositories.
Case study: **Freedom Toaster**				
	Ensured software is available offline for installation.	Enabled users to install and use software without Internet access.	Necessary for incorporating new software versions and updates.	Maintained consistency of software across devices.
Case study: **Navigation and Mapping**				
	Made maps available offline for continuous use.	Allowed for uninterrupted usage of navigation and mapping features without connectivity.	Critical for maintaining accuracy and relevance of maps over time.	Ensured data consistency between local and remote sources to keep maps up to date.
Case study: **Data Collection**				
	Provided initial data setup for offline data gathering.	Essential for collecting data in environments without Internet access.	Enabled synchronisation of collected data once connectivity is restored.	Ensured eventual consistency of collected data with remote databases.

In the GitHub Desktop case, the *Update* sub-pattern played a crucial role in maintaining

synchronisation between local and remote repositories, mirroring the ever-changing nature of software development workflows. The Freedom Toaster project, which aimed to distribute software in regions with intermittent Internet access, heavily depended on the *Post-Populate* sub-pattern to guarantee that users would receive the most up-to-date software versions upon reconnecting.

Navigation and mapping applications need to fully implement *all sub-patterns* to ensure that maps are preloaded, available offline, and regularly updated for accuracy. Conversely, the data collection use case prioritises *Offline Operation* and *Updates*, necessitating robust data collection mechanisms that can operate without Internet access and synchronise collected data upon restoring connectivity.

Similarly, the iNethi project emphasised *Pre-Population* and *Offline Operations* to offer Internet-like services in areas with unreliable connectivity, using *Post-Population* to ensure data consistency upon reconnection, and omitting the update sub-pattern due to the static content nature. The cases of GitHub Desktop and Freedom Toaster both emphasised the importance of *Pre-Population* and *Offline Operations* for initial setup and seamless functionality. The *Update* sub-pattern was crucial for synchronisation, while *Post-Population* ensured consistency. Navigation and mapping relied on all sub-patterns to ensure continuous offline map usage, accuracy, and relevance. In data collection, *Offline Operation* for data gathering without Internet access is essential, utilising *Updates* for synchronisation and *Post-Population* for consistency with remote databases. These case studies demonstrate the adaptability and effectiveness of the offline pattern and its sub-patterns in addressing specific requirements and limitations in low-resource environments.

6 RESULTS AND DISCUSSION

The proposed offline pattern was evaluated through six distinct case studies: OAI-PMH, iNethi, GitHub Desktop, Freedom Toaster, navigation and mapping, and data collection. These cases were carefully selected to represent various low-resource environment scenarios and to showcase the flexibility of the pattern. The results indicate that the offline pattern significantly improves the reliability and functionality of software systems in such environments. Each case study emphasised the pattern's effectiveness in data management and its ability to sustain operations despite intermittent or limited network connectivity. The consistent application of the pattern across different scenarios underscores its strength and adaptability.

An analysis of the case studies reveals the diverse applications and effectiveness of the offline pattern and its sub-patterns. For example, the Pre-Populate and Offline Operation sub-patterns were universally essential, enabling initial data availability and uninterrupted functionality, respectively. The Update sub-pattern demonstrated its value, particularly in scenarios such as GitHub Desktop and data collection, where real-time synchronisation was less critical but periodic updates were necessary. The Post-Populate sub-pattern proved notably beneficial in cases such as Freedom Toaster and iNethi, where offline-collected data needed to be reliably uploaded once connectivity was restored. This comparative insight highlights the

pattern's adaptability to address specific needs within different low-resource settings.

The strengths of the proposed offline pattern lie in its modularity and scalability. Decomposing the pattern into four distinct sub-patterns allows for tailored applications based on specific requirements, enhancing its utility across various domains. However, one limitation is its reliance on local storage, which could be restrictive in very resource-constrained environments. Furthermore, the pattern's effectiveness is closely tied to the initial setup and configuration, requiring thorough planning and implementation to fully realise its potential. Despite these limitations, the pattern represents a notable advancement in data management and ensuring operational continuity in low-resource scenarios.

The study suggests several recommendations for software practitioners. Firstly, software practitioners should thoroughly assess the operational environment to determine the most suitable sub-pattern combination. prioritising the Pre-Populate and Offline Operation sub-patterns can establish essential functionality. In contrast, the Update and Post-Populate sub-patterns can be applied selectively to address specific data synchronisation and upload needs. Practitioners should also account for potential constraints on local storage and develop strategies to address any limitations. Lastly, continual monitoring and iterative refinement of the implementation are essential to adapting the pattern to evolving requirements and improving its effectiveness in low-resource environments.

7 CONCLUSION

In low-resource environments, software systems often face limitations in processing power, memory, and network connectivity, making reliance on cloud storage challenging. Despite the use of strategies such as MQTT, CoAP, and caching to mitigate these issues, they remain dependent on network connectivity. In these contexts, the offline pattern introduced in this paper leverages local storage to enhance the functionality of software systems. By focusing on local storage, the offline pattern improves system efficiency and reliability, particularly for low-end devices. This distinction between offline storage and synchronisation is crucial, as it ensures that systems can maintain functionality independently of network availability, thus enhancing their reliability and usability in varied and often challenging conditions.

For the offline pattern to be implemented effectively, software developers should identify key functionalities that benefit from this approach, especially in low-resource settings. During design, pre-populating local storage with essential data is crucial to ensure seamless operation without network access. Emphasis should be placed on offline operation, minimising the need for real-time synchronisation. Regular updates are necessary to keep local storage synchronised with external data sources, facilitating smooth transitions between online and offline modes. Employ post-population techniques to maintain consistency between local and remote data, ensuring accurate data integration upon connectivity restoration. Future research could delve deeper into optimising local memory usage within the offline pattern, addressing storage challenges in hardware-constrained environments. Investigating the scalability and flexibility of the offline pattern in various settings will also be important to understand its

broader applicability.

ACKNOWLEDGEMENT

This research was partially funded by the *National Research Foundation* of South Africa (Grant numbers: 105862, 119121 and 129253) and the *University of Cape Town*. The authors acknowledge that the opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and that the NRF accepts no liability whatsoever in this regard.

References

Abdellatif, M., Hecht, G., Mili, H., Elboussaidi, G., Moha, N., Shatnawi, A., Privat, J., & Guéhéneuc, Y. (2018). State of the practice in service identification for SOA migration in industry. In C. Pahl, M. Vukovic, J. Yin & Q. Yu (Eds.), *Service-oriented computing* (pp. 634–650). Springer International Publishing. https://doi.org/10.1007/978-3-030-03596-9_46

Abou-Khalil, V., Helou, S., Khalifé, E., Chen, M. A., Majumdar, R., & Ogata, H. (2021). Emergency online learning in low-resource settings: Effective student engagement strategies. *Education Sciences*, *11*(1). https://doi.org/10.3390/educsci11010024

Akhigbe, B. I., Mtombeni, K., Densmore, M., & Suleman, H. (2022). Do people in low resource environments only need search? Exploring digital archive functionalities in South Africa. *Proceedings of 43rd Conference of the South African Institute of Computer Scientists and Information Technologists*, *85*, 158–171. https://pubs.cs.uct.ac.za/id/eprint/1532

Barakat, N. H. (2019). A framework for integrating software design patterns with game design framework. *Proceedings of the 8th International Conference on Software and Information Engineering*, 47–50. https://doi.org//10.1145/3328833.3328871

Biørn-Hansen, A., Majchrzak, T. A., & Grønli, T. (2018). Progressive web apps for the unified development of mobile applications. In T. A. Majchrzak, P. Traverso, K. Krempels & V. Monfort (Eds.), *Web information systems and technologies* (pp. 64–86). Springer International Publishing. https://doi.org/10.1007/978-3-319-93527-0_4

Bird, S., Hanke, F. R., Adams, O., & Lee, H. (2014). Aikuma: A mobile app for collaborative language documentation. *Proceedings of the 2014 Workshop on the Use of Computational Methods in the Study of Endangered Languages*, 1–5. https://aclanthology.org/W14-2201/

Bormann, C., Castellani, A. P., & Shelby, Z. (2012). CoAP: An application protocol for billions of tiny internet nodes. *IEEE Internet Computing*, *16*(2), 62–67. https://doi.org/10.1109/MIC.2012.29

Brahma, J., & Sadhya, D. (2022). Preserving contextual privacy for smart home IoT devices with dynamic traffic shaping. *IEEE Internet of Things Journal*, *9*(13), 11434–11441. https://doi.org/10.1109/JIOT.2021.3126453

Brunette, W., Sudar, S., Sundt, M., Larson, C., Beorse, J., & Anderson, R. (2017). Open Data Kit 2.0: A services-based application framework for disconnected data management. *MobiSys 2017: Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, 440–452. https://doi.org/10.1145/3081333.3081365

Brunette, W., Sundt, M., Dell, N., Chaudhri, R., Breit, N., & Borriello, G. (2013). Open Data Kit 2.0: Expanding and refining information services for developing regions. *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*. https://doi.org/10.1145/2444776.2444790

Buschmann, F., Henney, K., & Schmidt, D. C. (2007). *Pattern-oriented software architecture, a pattern language for distributed computing* (Vol. 4). John Wiley & Sons.

Cámara, J., Correia, P., de Lemos, R., & Vieira, M. (2014). Empirical resilience evaluation of an architecture-based self-adaptive software system. *Proceedings of the 10th International ACM Sigsoft Conference on Quality of Software Architectures*, 63–72. https://doi.org/10.1145/2602576.2602577

Chandwani, R., & Kumar, N. (2018). Stitching infrastructures to facilitate telemedicine for low-resource environments. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 1–12. https://doi.org/10.1145/3173574.3173958

Copps, E., Zhang, H., Sim, A., Wu, K., Monga, I., Guok, C., Würthwein, F., Davila, D., & Fajardo, E. (2021). Analyzing scientific data sharing patterns for in-network data caching. *SNTA 2021 – Proceedings of the 2021 Systems and Network Telemetry and Analytics, co-located with HPDC 2021*, 9–16. https://doi.org/10.1145/3452411.3464441

Demers, A., Petersen, K., Spreitzer, M., Terry, D., Theimer, M., & Welch, B. (1994). The Bayou architecture: Support for data sharing among mobile users. *1994 First Workshop on Mobile Computing Systems and Applications*, 2–7. https://doi.org/10.1109/WMCSA.1994.37

Durner, D., Chandramouli, B., & Li, Y. (2021). Crystal: A unified cache storage system for analytical databases. *Proceedings of the VLDB Endowment*, *14*(11), 2432–2444. https://doi.org/10.14778/3476249.3476292

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Pearson Deutschland GmbH.

Haque, S. R., Saha, S. S., Chowdhury, H. A., Ferdous, T. R., Chowdhury, A. S., Shihab, R. H., Choudhury, J. N., Choudhury, D. A. F. K., & Rahman, T. (2021). A cost-effective solution for real time remote monitoring of vital signs in patients. *2021 14th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, 1–7. https://doi.org/10.1109/CISP-BMEI53629.2021.9624436

Harjono, J., Ng, G., Kong, D., & Lo, J. (2010). Building smarter web applications with HTML5. *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research*, 402–403. https://doi.org/10.1145/1923947.1924015

Hartung, C., Lerer, A., Anokwa, Y., Tseng, C., Brunette, W., & Borriello, G. (2010). Open data kit: Tools to build information services for developing regions. *Proceedings of the 4th*

ACM/IEEE International Conference on Information and Communication Technologies and Development. https://doi.org/10.1145/2369220.2369236

Herrero, R. (2020). Analysis of the constrained application protocol over quick UDP internet connection transport. *Internet of Things*, *12*, 100328. https://doi.org/10.1016/j.iot.2020.100328

Hunkeler, U., Truong, H. L., & Stanford-Clark, A. (2008). MQTT-S — a publish/subscribe protocol for wireless sensor networks. *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)*, 791–798. https://doi.org/10.1109/COMSWA.2008.4554519

Hussain, W., Mougouei, D., & Whittle, J. (2018). Integrating social values into software design patterns. *Proceedings of the International Workshop on Software Fairness*, 8–14. https://doi.org/10.1145/3194770.3194777

Jannes, K., Lagaisse, B., & Joosen, W. (2021). OWebSync: Seamless synchronization of distributed web clients. *IEEE Transactions on Parallel and Distributed Systems*, *32*(9), 2338–2351. https://doi.org/10.1109/TPDS.2021.3066276

Jebessa, N. D., & Alemayehu, H. G. (2009). Mobile services and ICT4D to the network economy – bridging the digital divide, Ethiopia's case. *3rd Scientific Conference on Electrical Engineering, Organized by Ethiopian Society of Electrical Engineers and Electrical and Computer Engineering Department, Addis Ababa University*. https://doi.org/10.48550/arXiv.1401.7435

Jiang, S., & Mu, H. (2011). Design patterns in object oriented analysis and design. *2011 IEEE 2nd International Conference on Software Engineering and Service Science*, 326–329. https://doi.org/10.1109/ICSESS.2011.5982229

Kistler, J. J., & Satyanarayanan, M. (1992). Disconnected operation in the Coda File System. *ACM Transactions on Computer Systems*, *10*(1), 3–25. https://doi.org/10.1145/146941.146942

Kutoane, M., Brysiewicz, P., & Scott, T. (2021). Interventions for managing professional isolation among health professionals in low resource environments: A scoping review. *Health Science Reports*, *4*(3), e361. https://doi.org/10.1002/hsr2.361

Lagoze, C., & Van de Sompel, H. (2003). The making of the Open Archives Initiative protocol for metadata harvesting. *Library Hi Tech*, *21*(2), 118–128. https://doi.org/10.1108/07378830310479776

Leimeister, J. M., Dickhaut, E., & Janson, A. (2021). Design pattern as a bridge between problem-space and solution-space. In S. Aier, P. Rohner & J. Schelp (Eds.), *Engineering the transformation of the enterprise: A design science research perspective* (pp. 137–150). Springer International Publishing. https://doi.org/10.1007/978-3-030-84655-8_9

Liu, W. T. (2014). Research on offline storage of web page. *Applied Mechanics and Materials*, *518*, 305–309. https://doi.org/10.4028/www.scientific.net/AMM.518.305

Maddah-Ali, M. A., & Niesen, U. (2014). Fundamental limits of caching. *IEEE Transactions on Information Theory*, *60*(5), 2856–2867. https://doi.org/10.1109/TIT.2014.2306938

Meheden, M., Musat, A., Traciu, A., Viziteu, A., Onu, A., Filote, C., & Răboacă, M. S. (2021). Design patterns and electric vehicle charging software. *Applied Sciences*, *11*(1). https://doi.org/10.3390/app11010140

Mertz, J., Nunes, I., Della Toffola, L., Selakovic, M., & Pradel, M. (2021). Satisfying increasing performance requirements with caching at the application level. *IEEE Software*, *38*(3), 87–95. https://doi.org/10.1109/MS.2020.3033508

Nungu, A., & Pehrson, B. (2011). Towards sustainable broadband communication in rural areas. In R. Szabó, H. Zhu, S. Imre & R. Chaparadza (Eds.), *Access networks* (pp. 168–175, Vol. 63). Springer Berlin Heidelberg, Lecture Notes of the Institute for Computer Sciences, Social Informatics; Telecommunications Engineering. https://doi.org/10.1007/978-3-642-20931-4_13

Pentland, A., Fletcher, R., & Hasson, A. (2004). DakNet: Rethinking connectivity in developing nations. *Computer*, *37*(1), 78–83. https://doi.org/10.1109/MC.2004.1260729

Phiri, L., Williams, K., Robinson, M., Hammar, S., & Suleman, H. (2012). Bonolo: A general digital library system for file-based collections. In H. Chen & G. Chowdhury (Eds.), *The outreach of digital libraries: A globalized resource network* (pp. 49–58). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-34752-8_6

Phokeer, A., Hadzic, S., Nitschke, E., Van Zyl, A., Johnson, D., Densmore, M., & Chavula, J. (2020). iNethi community network: A first look at local and internet traffic usage. *Proceedings of the 3rd ACM SIGCAS Conference on Computing and Sustainable Societies*, 342–344. https://doi.org/10.1145/3378393.3402289

Quinn, H., Baker, Z., Fairbanks, T., Tripp, J. L., & Duran, G. (2015). Software resilience and the effectiveness of software mitigation in microcontrollers. *IEEE Transactions on Nuclear Science*, *62*(6), 2532–2538. https://doi.org/10.1109/TNS.2015.2496342

Ramanujapuram, A., & Malemarpuram, C. K. (2020). Enabling sustainable behaviors of data recording and use in low-resource supply chains. *Proceedings of the 3rd ACM SIGCAS Conference on Computing and Sustainable Societies*, 65–75. https://doi.org/10.1145/3378393.3402248

Rasool, G., & Akhtar, H. (2019). Towards a catalog of design patterns variants. *2019 International Conference on Frontiers of Information Technology (FIT)*, 156–161. https://doi.org/10.1109/FIT47737.2019.00038

Rath, A., Spasic, B., Boucart, N., & Thiran, P. (2019). Security pattern for cloud SaaS: From system and data security to privacy case study in AWS and Azure. *Computers*, *8*(2). https://doi.org//10.3390/computers8020034

Satyanarayanan, M. (1989). Coda: A highly available file system for a distributed workstation environment. *Proceedings of the Second Workshop on Workstation Operating Systems*, 114–116. https://doi.org/10.1109/WWOS.1989.109279

Satyanarayanan, M., Kistler, J., Kumar, P., Okasaki, M., Siegel, E., & Steere, D. (1990). Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, *39*(4), 447–459. https://doi.org/10.1109/12.54838

Scott, C. (2016). Crowd powered media delivery: Facilitating ubiquitous device-to-device file transfers. *Proceedings of the 7th Annual Symposium on Computing for Development*. https://doi.org/10.1145/3001913.3006640

Seidl, C., Schuster, S., & Schaefer, I. (2017). Generative software product line development using variability-aware design patterns. *Computer Languages, Systems & Structures*, *48*, 89–111. https://doi.org/10.1016/j.cl.2016.08.006

Singh, N., & Hasan, M. (2019). Efficient method for data synchronization in mobile database. *2019 IEEE Conference on Information and Communication Technology*, 1–5. https://doi.org/10.1109/CICT48419.2019.9066122

Suleman, H. (2004). NDLTD union catalog project. In *Electronic theses and dissertations* (pp. 89–92). CRC Press. https://www.taylorfrancis.com/chapters/edit/10.4324/9780203021279-14/ndltd-union-catalog-project-hussein-suleman

Suleman, H. (2019). Reflections on design principles for a digital repository in a low resource environment. *Proceedings of HistoInformatics Workshop 2019*, 1–10. http://pubs.cs.uct.ac.za/id/eprint/1331

Suleman, H. (2021). Simple DL: A toolkit to create simple digital libraries. In H. Ke, C. S. Lee & K. Sugiyama (Eds.), *Towards open and trustworthy digital societies* (pp. 325–333). Springer International Publishing. https://doi.org/10.1007/978-3-030-91669-5_25

Suleman, H. (2022). Investigating evolving collection support with simple tools. In Y. Tseng, M. Katsurai & H. N. Nguyen (Eds.), *From born-physical to born-virtual: Augmenting intelligence in digital libraries* (pp. 449–455). Springer International Publishing. https://doi.org/10.1007/978-3-031-21756-2_36

Surana, S., Patra, R., Nedevschi, S., & Brewer, E. (2008). Deploying a rural wireless telemedicine system: Experiences in sustainability. *Computer*, *41*(6), 48–56. https://doi.org/10.1109/MC.2008.184

Tang, Y., Dananjayan, S., Hou, C., Guo, Q., Luo, S., & He, Y. (2021). A survey on the 5G network and its impact on agriculture: Challenges and opportunities. *Computers and Electronics in Agriculture*, *180*, 105895. https://doi.org/10.1016/j.compag.2020.105895

Terry, D. B., Theimer, M. M., Petersen, K., Demers, A. J., Spreitzer, M. J., & Hauser, C. H. (1995). Managing update conflicts in Bayou, a weakly connected replicated storage system. *ACM SIGOPS Operating Systems Review*, *29*(5), 172–182. https://doi.org/10.1145/224057.224070

Thangavel, D., Ma, X., Valera, A., Tan, H., & Tan, C. K. (2014). Performance evaluation of MQTT and CoAP via a common middleware. *2014 IEEE 9th International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, 1–6. https://doi.org/10.1109/ISSNIP.2014.6827678

Tridgell, A., & Mackerras, P. (1996). *The RSync algorithm* (tech. rep.). The Australian National University. https://openresearch-repository.anu.edu.au/items/15a1c428-0ad3-49d6-bb54-9238250cbbf0

USDA. (2019). *A case for rural broadband: Insights on rural broadband infrastructure and next generation precision agriculture technologies* (tech. rep.). United States Department of Ag-

riculture. https://www.usda.gov/sites/default/files/documents/case-for-rural-broadband.pdf

Vanhala, J. (2017). *Implementing an offline first web application* [Master's thesis]. Aalto University. School of Science. https://aaltodoc.aalto.fi/handle/123456789/29096

von Holy, A., Bresler, A., Shuman, O., Chavula, C., & Suleman, H. (2017). Bantuweb: A digital library for resource scarce South African languages. *Proceedings of the South African Institute of Computer Scientists and Information Technologists*. https://doi.org/10.1145/3129416.3129446

Wang, J. (1999). A survey of web caching schemes for the internet. *ACM SIGCOMM Computer Communication Review*, *29*(5), 36–46. https://doi.org/10.1145/505696.505701

Wang, Z. (2022). IDFS: The individual distributed file system. In W. Wei (Ed.), *International conference on automation control, algorithm, and intelligent bionics (ACAIB 2022)* (pp. 352–360, Vol. 12253). SPIE. https://doi.org/10.1117/12.2639431

Xu, P., Zhao, N., Wan, J., Liu, W., Chen, S., Zhou, Y., Albahar, H., Liu, H., Tang, L., & Tan, Z. (2022). Building a fast and efficient LSM-tree store by integrating local storage with cloud storage. *ACM Transactions on Architecture and Code Optimization*, *19*(3). https://doi.org/10.1145/3527452

Zhang, N., Guo, S., Dong, Y., & Liu, D. (2020). Joint task offloading and data caching in mobile edge computing networks. *Computer Networks*, *182*, 107446. https://doi.org/10.1016/J.COMNET.2020.107446