Data and Metadata. 2024; 3:.225
doi: 10.56294/dm2024.225

REVIEW

The impact of distributed systems on the architecture and design of computer systems: advantages and challenges

El impacto de los sistemas distribuidos en la arquitectura y el diseño de sistemas informáticos: ventajas y retos

Yevhenii Tytarchuk[1] ✉, Sergii Pakhomov[2], Dmytro Beirak[3], Vasyl Sydorchuk[3], Svitlana Vasylyuk-Zaitseva[4]

[1]Department of Computer Science and Digital Economy, Faculty of Economics, Information Technology and Service, Vinnytsia National Agrarian University, Ukraine.
[2]Department of Cybernetics of Chemical Technology Processes, Faculty of Chemical Technology, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Ukraine.
[3]Department of Software Engineering, Zhytomyr Polytechnic State University, Ukraine.
[4]Computer Science Department, Information Technologies Faculty, National University of Life and Environmental Sciences of Ukraine, Ukraine.

Cite as: Tytarchuk Y, Pakhomov S, Beirak D, Sydorchuk V, Vasylyuk Zaitseva S. The impact of distributed systems on the architecture and design of computer systems: advantages and challenges. Data and Metadata. 2024;3:.225. https://doi.org/10.56294/dm2024.225

Submitted: 10-02-2024 **Revised:** 14-07-2024 **Accepted:** 27-12-2024 **Published:** 28-12-2024

Editor: Adrián Alejandro Vitón Castillo

Corresponding Author: Yevhenii Tytarchuk[1] ✉

ABSTRACT

A distributed system can encompass a variety of configurations, including mainframes, personal computers, workstations, and minicomputers. The varying degrees of software flexibility and the ability to execute tasks in parallel facilitate simultaneous data processing across multiple processors. The higher the resilience of an application, the quicker it can recover after a system failure. Organisations increasingly adopt distributed computing systems as they face increased data generation and demand for enhanced application performance. These systems enable businesses to scale effectively in response to growing data volumes. Integrating additional hardware into a distributed system is generally simpler than upgrading a centralised system reliant on powerful servers. Distributed systems comprise numerous nodes that collaborate towards a common objective. This article aims to provide a comprehensive overview of distributed systems, their architectural frameworks, and essential components. This study examines how distributed systems influence the architecture and design of computer systems. The research methods consist of reviewing existing literature and analysing case studies on implementing distributed systems. Key findings indicate that the evolution of distributed systems is ongoing, driven by emerging technologies and the increasing demand for efficient, scalable, and secure solutions. Innovations such as edge computing, blockchain technology, 5G, and the integration of AI and machine learning are among the notable trends shaping the future landscape of distributed systems. Looking ahead, designers and architects need to stay informed about these advancements to create reliable and adaptable distributed systems that can address the dynamic needs of users and organisations.

Keywords: Distributed System; Fault Tolerance; Performance; Scalability; System Architecture.

RESUMEN

Un sistema distribuido puede abarcar una gran variedad de configuraciones, como mainframes, ordenadores personales, estaciones de trabajo y miniordenadores. Los distintos grados de flexibilidad del software y la capacidad de ejecutar tareas en paralelo facilitan el procesamiento simultáneo de datos en varios procesadores. Cuanto mayor sea la resiliencia de una aplicación, más rápido podrá recuperarse tras un fallo

del sistema. Las organizaciones adoptan cada vez más sistemas informáticos distribuidos ante el aumento de la generación de datos y la demanda de un mayor rendimiento de las aplicaciones. Estos sistemas permiten a las empresas escalar eficazmente en respuesta a los crecientes volúmenes de datos. Integrar hardware adicional en un sistema distribuido suele ser más sencillo que actualizar un sistema centralizado basado en potentes servidores. Los sistemas distribuidos están formados por numerosos nodos que colaboran en pos de un objetivo común. Este artículo pretende ofrecer una visión global de los sistemas distribuidos, sus marcos arquitectónicos y sus componentes esenciales. Este estudio examina cómo influyen los sistemas distribuidos en la arquitectura y el diseño de los sistemas informáticos. Los métodos de investigación consisten en revisar la bibliografía existente y analizar casos prácticos de implantación de sistemas distribuidos. Las principales conclusiones indican que la evolución de los sistemas distribuidos es continua, impulsada por las tecnologías emergentes y la creciente demanda de soluciones eficientes, escalables y seguras. Innovaciones como el edge computing, la tecnología blockchain, el 5G y la integración de la IA y el aprendizaje automático son algunas de las tendencias destacadas que configuran el panorama futuro de los sistemas distribuidos. De cara al futuro, los diseñadores y arquitectos deben mantenerse informados sobre estos avances para crear sistemas distribuidos fiables y adaptables que puedan responder a las necesidades dinámicas de los usuarios y las organizaciones.

Palabras clave: Sistema Distribuido; Tolerancia a Fallos; Rendimiento; Escalabilidad; Arquitectura de Sistemas.

INTRODUCTION

The distribution system and network provide the communication, computing, and data storage infrastructure, potentially reaching many worldwide seeking the same information, hardware devices, and software processes. The main concerns are scalability, performance, functionality and manageability. Distributed networks are widely used in computer programming, software processing, and distributed computing, where data and databases are distributed among multiple computers. Still, complex messages are sent through a source node (computer) and are mostly interconnected. The purpose of a distributed network is to share resources on a global scale, often to achieve separate or similar visual goals. Distributed network systems consist of flows, agents, processes, and uniformly distributed objects.[1,2,3]

There are two operating systems for distributed systems: distributed and networked. The system is distributed among all nodes in the first case, so the network is practically hidden from the user. In the second case, each node contains an exact copy of the system in a network operating system, and the network itself is invisible.

This article explores how distributed systems influence the architecture and design of computer systems and their associated benefits and drawbacks. It also examines the concept of distributed architecture and evaluates the advantages and disadvantages of implementing distributed systems within that framework.

One of the primary and dominant features of humanity's development is the globalisation of all spheres of its activity, including political, social, economic, ecological, scientific, and technical.

The common denominator of such processes' diversity is information flows from various databases. The most common are vast volumes of digital, text, graphic, audio, and video information, which dynamically change and qualitatively transform.

The specified information bases make up the input arrays of data for processing with modern computer equipment.

An essential aspect of such processing is the presence of powerful high-speed processors and a complex of modern algorithmic approaches to intelligent analysis of information databases.[4]

METHOD

The methodology involves conducting a literature review and analysing case studies on distributed systems. While writing this article, the literature review revealed that distributed systems are highly significant in the current technological landscape. Examples of the use of distributed systems were also considered; for example, distributed systems allow Amazon Web Services (AWS) to manage EC2 instances using a set of distributed algorithms and mechanisms, such as data replication, load balancing, automatic scaling and load balancing. For example, AWS uses ELB (Elastic Load Balancing) to distribute traffic between EC2 instances in different availability zones.

The methodology involves a thorough literature review complemented by the analysis of real-world case studies. For example, AWS utilizes distributed algorithms like data replication, load balancing, and automatic scaling to enhance service delivery. Tools such as Elastic Load Balancing (ELB) effectively distribute traffic across multiple EC2 instances, showcasing the practical benefits of distributed systems.

RESULTS AND DISCUSSION

This feature is a direct consequence of the existence of independent computers. Still, at the same time, it does not show how these computers are connected to a single system. Distributed systems are usually always available, but some parts may fail temporarily. Users and applications should not be notified that these parts will be replaced or repaired or that new parts will be added to support additional users or applications.

A distributed system configuration usually includes an additional software layer between the upper layer containing users and applications and the lower layer of the operating system to support the representation of different computers and networks as a single system. Therefore, such distributed systems are often referred to as middleware systems.

The primary challenge addressed by distributed systems technology is enabling access to resources spread across the globe and tackling computational tasks that demand substantial processing power beyond the capabilities of conventional computers. Implementing global tasks is complex because different computers can access the necessary data. In addition, distributed systems consisting of autonomous resources can dynamically change their architecture. The developer's task of a distributed system is to design software and hardware so that all functions necessary for a distributed system can be implemented.

In distributed systems, various components can be developed using different programming languages and operate on distinct processors. The data models, information formats, and communication protocols employed in these systems do not need to be uniform. Therefore, distributed systems require software that manages these different components and allows them to interact and exchange data. Middleware belongs to this class of software. Middleware sits in the middle between the various components of a distributed system.

What is a distributed architecture?

Distributed systems are created based on existing network software and operating systems. A distributed system comprises autonomous computers connected through a network and distribution middleware. To become autonomous, there is a clear master/slave relationship between two computers on a network. Middleware allows computers to coordinate their activities and share system resources so that users perceive the system as a single, integrated computing facility. Thus, middleware is a bridge that connects distributed applications in different physical locations with various hardware platforms, network technologies, operating systems, and programming languages. Middleware software is developed according to agreed standards and protocols. It provides standard services such as naming, persistence, concurrency control to ensure accurate results for parallel processes and results as quickly as possible, event distribution, authorisation to determine resource access rights, security, etc.[5,6]

Because the performance of distributed systems depends in part on these system configuration parameters, there is growing interest in how best to use these configuration parameters to improve system performance for three reasons:

1. Parameter values are easy to configure because they are written as plain or XML text, commonly used in mainstream servers such as Apache, JBoss, and MySQL.
2. Parameter configuration requires little additional cost compared to hardware investment.
3. Parameter values significantly impact system performance. Tests show that the optimal configuration can increase system throughput by 24,5 % and reduce average response time by 28,1 % compared to the default configuration.[7,8]

Advantages of using distributed systems in architecture

The transparency of the distribution reflects the degree to which the system is perceived as a whole rather than as a set of independent components. In other words, the higher the transparency, the more the user can see the system as a whole. The extreme case is when achieving a unified view of the system is possible, which does not differ from the disaggregated system.

The problem with achieving allocation transparency in extensive systems is that performance can drop to unacceptable levels. The reasons for this are apparent. Because the network is interconnected, failures and recoveries that cannot be fully masked must be handled. In addition, there is a natural lower limit to network latency, which becomes more noticeable when working with long-distance connections.

These performance problems can be partially solved by replication, where components are copied and placed close to their intended location. Still, since the read-update rate is reduced, replication reduces the size of the system when consistency is required.[9] In addition, the CAP principle teaches that consistency and availability cannot simply be combined in the presence of network partitions.[10]

There is no simple solution to achieving distribution transparency, and application-specific solutions must continue to be sought to attain acceptable transparency. The last word has not yet been said, and the challenge is finding something adequate for users.

Another challenge for large-scale distributed systems is working with the Internet of Things. The Internet

of Things is the ubiquity of many IP-enabled objects, from tags attached to products to mobile devices and services.[11] From a distributed systems perspective, the challenge is to move away from network- and things-centric views and provide a perspective in which the collaboration of all these things, with the support of the Internet of Things, actually forms a complete distributed system.

Essentially, we mean the heterogeneity inherent in today's large-scale distributed systems.

To achieve this perspective, it is necessary to solve significant problems:

How can such subsystems as sensor networks be designed as distributed systems? For example, work is underway to consider such networks as distributed databases[12] or to provide tools for programming them as natural systems.[13]

Given the potential volume of data generated by sensors and connected devices, a balance must be struck between processing and aggregation within the network and streaming data to an external support system. Achieving this balance is not easy. It depends on the capabilities of the distributed sensor network, the communication channels between the sensor network and the support system, and the support system itself. Again, the sheer size of such systems (think, for example, hundreds of thousands of mobile devices acting as sensors) can be daunting.

Many modern distributed systems must be located in multiple management organisations. Indeed, the development of collaborative systems is almost inevitable if we are to maintain large-scale decentralisation. To put it mildly, we argue that a deep understanding of the construction of such a system, for example, essential security and reliability requirements, is lacking. In fact, as the security challenges of structured peer-to-peer networks[14] show, we should ask ourselves if this is even possible. Problems related to cooperation in large-scale distributed systems are not new and have been considered by researchers in the (algorithmic) design of mechanisms and subfields of game theory. Developing mechanisms that work, scale, and can be effectively implemented is challenging. We anticipate it will take a lot of effort before a generally accepted solution emerges. The success of BitTorrent demonstrates that we are not faced with an unsolvable problem.

As mentioned above, many challenges in developing distributed systems are well-known or prominent. However, if we look around, we will see that more problems need to be solved, and research programs need to begin to be coordinated.

Let's look at one of the crucial developments happening now: the emergence of sociotechnical systems. These systems integrate people and computer systems (including hardware and software) and their (probably mostly implicit) rules of use and interaction. The key word here is integration. In sociotechnical systems, the boundaries between people, technology, and use are blurred.

In this context, many distributed systems are designed to be used and shared by people, but the users and the system are more or less separated. There are many systems for casual user-to-user communication, for example, based on (mobile) phones and various user-oriented messaging systems.[15,16,17] In addition, the last decade has seen an explosion of web-based systems that provide end-user services via a web browser, such as online trading systems, e-commerce systems, and distributed systems that implement information services. More recently, as predicted by Mark Weiser,[18] ubiquitous computerisation[19] is becoming a reality. For example, many public transport systems are now equipped with RFID-based Ticketing, and supermarkets can scan your products for shoppers.

As mentioned above, in the design of these distributed systems, the end users are considered disconnected from the systems; that is, they choose the system. As a result, many efforts have always been reported to increase the transparency of distribution by hiding, where appropriate, the data, processes, and controls deployed in the system. We believe that future large-scale systems will encounter significant challenges, after which they will become more scalable.

To illustrate what the problem is, consider the following scenario. Bob likes to listen to music and is interested in things he has never heard of. His musical taste is very diverse and covers many different genres of music, so to keep up to date, he subscribes to various periodicals and recommends reference records. Using these and other sources, Bob purchases CDs, purchases songs from multiple online resources, and uses audio streaming services such as Spotify.[20]

If we go even further, the future scenario in which users and distributed systems will be trapped in a vicious circle may look like this.

Bob is given a personalised e-zine downloaded onto his tablet. This magazine does not differentiate between musical styles. It is personalised, so there is no need to distinguish between them. The system knows what information to send Bob because it has learned to make recommendations by observing what he has listened to and bought. The system has also worked out the best way to introduce new material that is likely to be most appreciated. In addition to these recommendations, the system also includes the best search box for Bob (the size and content of which can depend on the current context) so that there is ample opportunity to efficiently and effectively find new content while browsing.

The system automatically determines which criticisms the user will react to (for example, which music

to buy). It is also effectively organised as a publish-subscribe system that constantly evolves with the help of content analysis.

But that's not all. In this example, the system automatically builds a social network around musical interests to improve the search process, connecting Bob with potential peers. First, this is done to increase the efficiency of content search. However, it is desirable to use a more complex decentralised evolutionary algorithm that optimises which attributes form meaningful connections.[21] In the latter case, this social network allows Bob to connect with others (if he chooses) to discuss music. But the most important thing is that over time, he will see a personalised library of his favourite music, which will be constantly replenished with new music that suits his taste and at a pace that suits his lifestyle.

It is important to note that the boundary between the user and the computer system is disappearing. In this example, Bob constantly responds to what the system has to offer, and the system continually responds to Bob. Moreover, Bob is not alone. The other users and their actions, be they listeners, commenters, or bloggers, make the system work.

Even just looking at this example scenario, you can run into many problems. This distributed system comprises several local media systems in the user's home. Regional systems should run as thin clients, but that doesn't matter. The whole system is based on a set of services that work in the cloud. A significant part of distributed systems research focused on developing such services, mainly for cluster computer systems. The challenge is to create additional components that can track user interactions, perform appropriate analysis, and adapt to individual use cases. In addition, many of the services offered must be integrated and distributed between cloud and on-premises media systems and meet performance criteria. In short, we argue that a socio-technical distributed system creates a feedback loop, and the main challenge is to close this loop effectively. Now, let's look at the different parts of this architecture.

Local media systems have many hardware and software components that can change over time. The goal is to simplify configuration. Hardware components include network audio/video devices and controls, e-book readers, notification devices, and secure payment devices. For example, many of these components can be integrated into modern tablets, separate smartphones, or specialised devices.

Device management is complex, but we've already come a long way using protocols like UPnP. However, as anyone who has installed a networked AV system can attest, there is still plenty of room for improvement. In addition, as people become increasingly mobile, they face complex challenges with network and content access. However, many of these hardware-related issues are being resolved and are not expected to become significant issues shortly.

In addition, managing the underlying software components is a reasonably well-managed technique. Most components can check for updates automatically; in most cases, the process can only be initiated with the user's consent. Although there is room for improvement and the science of system management and operation is undoubtedly not well developed,[22] we do not believe that traditional management is one of the biggest challenges.

If we want to create the kind of adaptive system we think we will need in the future; the main challenge is to accurately and effectively measure and monitor user behaviour. In this example, you need to collect the following statistics

- What music was purchased
- What music did you listen to, when, for how long and under what conditions?
- What information about music did you read, when and for how long?

This example can be complicated by the possibility of several people listening to the same music together. Similarly, in the case of video streaming systems, collective viewing of the film must be considered. In addition, assuming that the system also provides recommendations, user feedback on the quality of the recommendations is needed. This feedback can be implicit or directly requested by the user (by checking compliance with recommendations).

Of course, the main challenge in (decentralised) user analytics is finding the balance between privacy and advice/alerts. Without information from the user, it is almost impossible to create a system that optimally adapts to him. On the other hand, if user information becomes known and shared decentralised to improve the quality of advice and tailoring, it can quickly provide information that should remain confidential. This is a well-known problem,[23,24] but it is expected to remain one of the main challenges for large-scale socio-technical distributed systems. Next, we will consider two main problems: user involvement in system design and system management.

Many future challenges in distributed systems arise from the unpredictability and spontaneity that users bring when they become part of the system. In the previous example, the system needs to know how to work best for a single user, and that user essentially only needs their input. Future systems are expected to consider more users to achieve an acceptable level of performance. Indeed, Web 2.0, blogs and social networks

https://doi.org/10.56294/dm2024.225

transform users from passive information consumers to active players who shape services and define content and interaction.

The key word here is crowdsourcing, which means delegating tasks to an unspecified number of people through a public tender. Wikipedia is perhaps the most famous example of crowdsourcing, with almost 4 000 000 articles (an English-language site) written by more than 600 000 people.

For example, GWAP (Game With A Purpose) is a simple online game. Still, it has proven very effective in accurately and succinctly labelling an extensive collection of player photos.

In such systems, people are fully integrated into the system. The traditional model, where the computer performs the calculations, and the person only observes them, is completely reversed. The computer coordinates while a person performs the actual calculations. Common to these systems is an intelligent design that combines attractive incentives for voluntary human participation with mechanisms for filtering low-quality input data and collecting valuable data.

This increasing interdependence between seemingly independent subsystems creates new challenges for large-scale distributed systems. In this context, how can we guarantee correctness, fault tolerance, performance, privacy, etc.? To paraphrase Leslie Lamport's famous words, a very large distributed system is a system in which the failure of an external service that you didn't even know existed renders your application unusable.

A good example is the Amazon EC2 crash in April 2011.[25] This event was caused by a router misconfiguration, which prevented the normal operation of the system. However, this event forced many interconnected systems to take corrective measures. While each of these actions made sense when analysed individually, the cumulative effect of these actions was detrimental and exacerbated the problem rather than solving it. Here, the emergent behaviour of distributed systems in the conditions of complex networks is observed.[26]

Solving this critical problem requires predicting the impact of any management action, such as updating a configuration file or starting or stopping a particular computer. Recent examples have shown that even within the same management domain, the consequences of such actions are far from fully understood. Extending such an assessment to predict impacts on other external but related systems is a challenging research question that we need to address. Such forecasting is probably complicated because each system manager keeps some of the internal details of their system secret. It may be sufficient to observe whether an action has a positive or negative effect on the global system and to construct the system so that it can continuously move towards the desired state.

Fault tolerance management may also require effective mechanisms for detecting failed components in large distributed systems.

In other words, users enter the design arena and become first-class citizens of distributed systems. In doing so, they face severe administrative problems that administrators cannot solve. To take the next step, they may need to tap into the collective knowledge of these same users.

The answer may indeed lie in the creation of a system of self-governance. However, looking at how we're doing it now, it feels like we're running faster with more people on a growing but familiar trail. Maybe it's time to start running a different way.

Why don't we manage fewer people and let the system tell us what we should manage? Tuning the system is essential, but maybe it (fine tuning) should be put off for a while. Our work tests how to use a fully decentralised evolutionary algorithm to find the best way to cluster users semantically. We add parameters, and evolution works. The results are worse than when we developed the solution using our own experience, but still entirely satisfactory. This is already the beginning.

However, finding a solution that gives us more control over crash behaviour, perhaps without fully understanding how that solution works, can be an essential step we should take more often as a middleware and systems development community. Relinquishing control can be the hardest step. Also, finding and making solutions that usually work but never solve the problem may be something we should take more seriously; the work of Qin et al., presented at SOSP in 2005, can be an exciting starting point.[27]

While numerous definitions of distributed systems exist in the literature, none is perfectly satisfactory or universally consistent. For this discussion, a straightforward definition will suffice: a distributed system assembles independent computers that appear to the user as a cohesive system. This definition encompasses two aspects: the hardware dimension, where all computers function autonomously, and the software aspect, where the user perceives a unified system. Both dimensions are crucial. Rather than being preoccupied with definitions, it may be more beneficial to concentrate on the key characteristics of distributed systems: firstly, the variances among computers and their interaction methods are not apparent to the user. This holds for the external organisation of distributed systems as well.

In addition, distributed systems should be relatively easy to expand (scalable). This characteristic is a direct consequence of the existence of independent computers. Still, at the same time, it does not show how these computers are combined into a single system. A distributed system is usually always available, but some parts

may be temporarily unavailable. Users and applications should not be notified that these parts will be replaced or repaired or that new parts will be added to support additional users or applications.

The configuration of distributed systems often includes an additional layer of software between the upper user or application layer and the lower layers of the operating system to support the operation of different computers and networks as a single system. For this reason, such distributed systems are often called middleware systems. All distributed systems contain one or more processors, but how they are organised into systems differs. This especially applies to the way processors are connected and shared.

Virtual reality testing involves simulating the environment, user interactions, and implications of software development using VR hardware and software. It can be applied in education, gaming, healthcare, and technology. Virtual reality testing can assess a design's usability, usefulness, accessibility, and aesthetics, as well as users' emotional and cognitive responses.

The VR device includes a motion tracking system and a display that displays a virtual scenario to begin the VR testing process. Therefore, it is essential to introduce a VR program to design this virtual environment and its interaction in the digital space. This software is critical, especially for applications such as computer games, 3D modelling tools, and frameworks. In addition, a user testing methodology must be developed to obtain input and build a test script to guide the audience through the process. This can be achieved through questionnaires, surveys, interviews, or observations. A practical VR test certainly needs each of these elements to improve.

VR assessment provides new methods for predicting patient recovery and treatment outcomes. There are also innovations in virtual reality in orthopaedics, stroke rehabilitation, mental health and intensive care. In addition, virtual reality testing improves realism, reducing impulsive reactions and distractions. Finally, in general, VR has the potential to offer a more intensive form of rehabilitation that, if designed correctly, can be a replica of your software. Users will have the feeling of being in and interacting with the technology. Another feature of VR is that it will allow you to experience new features, UI/UX and functions that are not physically possible in the real world. Finally, it will provide enough data with highly targeted user input to refine your product design further.

In addition to cost, a VR experiment can have innovation and usability issues, ethical considerations, and potential risks. They are out of reach for the consumer and developer, who may be unable to avoid spending on virtual reality devices and software requiring complex hardware and software requirements. However, the expert creation of VR software cannot be trivial due to the complexity that requires careful 3D modelling, animation, programming and user interface design. Ethical issues regarding the use of VR for testing need to be addressed: permissions, privacy, security, and deception are topics that need to be addressed. In addition, users may experience disadvantages of VR testing, including fatigue, motion sickness, eye strain, and disorientation, so pre- and post-testing should be conducted for mitigation purposes.

Virtual reality, or immersive testing, is a new and vital method developers have adopted to test software performance. This is mostly about setting various conditions at the beginning so it can offer the feedback that users need and also create different design solutions. Perhaps most of the challenges are ones that the average person has to think about and strategise about, related to cost/complexity/ethics/side effects, among others. Before delving into virtual reality testing, it is essential to define clear goals. One crucial factor to consider is determining the most appropriate VR hardware and software. Creating a VR environment that matches the user's characteristics is very important. Using an effective user acceptance testing methodology is also important, as the method allows for collecting and analysing the user's response, which should be reliable and informative. Careful implementation of good clinical practice helps to achieve a positive result.[28]

The software ecosystem maturity stages described in the previous section are related to assessing the potential for ecosystem formation. The increase in the number of participants in software ecosystems and their activities affects the attractiveness of software ecosystems in the region and ensures their potential growth: the appearance and cooperation of new participants will lead to the creation of new companies, developments, and innovative business models, which will create a base for further development of the territory.

Currently, there are no tools to assess the maturity of software ecosystems, considering the potential for innovation in the region. Using existing methods for evaluating software ecosystem development is problematic due to the complexity of data collection. In addition, they are only focused on analysing the number of connections between participants, identifying hub participants (i.e., having the most significant number of connections), etc., without touching on the issues of power estimation.

A variety of activities are required for software ecosystems to work sustainably. The change in the number of species and their composition determines the ecosystem's development and affects the change in maturity stages. Thus, diversity is key to the existence and development of software ecosystems.[29]

CONCLUSIONS

The primary objective addressed by distributed systems technology is to facilitate global access to resources and handle tasks demanding substantial computational power that traditional computers cannot support.

The challenge in executing global tasks arises because various computers may have access to the required data. Furthermore, distributed systems, which consist of autonomous resources, can dynamically adjust their architecture. A distributed system can be described as a collection of independent computers linked by communication channels that appear to a specific software user as a single coherent entity. This topic is particularly significant and relevant in today's landscape of rapid and large-scale computing on computers.

Distributed systems consist of computers that operate independently but can be interconnected to create the appearance of a single connected system. The advantage is that different programs running on other computers can be easily integrated into a single system and, if properly designed, are highly scalable.

The size of a distributed system is limited only by the size of the underlying network. Distributed systems are classified according to principles based on hardware and software characteristics. The application of distributed systems is characterised by software complexity, low performance, and security issues. Distributed operating systems are used to manage interconnected computer systems' hardware and are considered a single system.

Network operating systems effectively connect different computers running their operating systems and provide users access to local services on each node. However, network operating systems do not give users the feeling of working in a single system, characteristic of distributed operating systems. Distributed systems have the following characteristics:

1. components of a distributed system are spatially distributed and interact locally or remotely;
2. components of a distributed system can work in parallel, which increases productivity compared to sequential work;
3. the state of each component is evaluated locally, that is, from the point of view of a specific computing process called from a local workstation.
4. distributed systems are affected by partial system failures since each component works separately and can "fail" independently of each other without interrupting the operation of the entire system;
5. systems work asynchronously, communication and processing processes are not controlled by the global system time, and variables and methods are synchronised;
6. in distributed systems, no single component can exercise all control, so control functions are distributed among different autonomous components to guarantee a certain degree of autonomy;
7. distributed systems can be created by combining existing systems. Complete contextual control of names is required, allowing for the same interpretation in administrative and technical domains.
8. Prospects for further research lie in the great potential for expanding the boundaries of what is possible and improving the functionality of distributed systems.

Distributed systems significantly influence modern computing, offering unparalleled scalability, reliability, and resource efficiency. However, achieving transparency, managing heterogeneity, and integrating socio-technical components remain pressing challenges. Continued research and innovation in middleware development, configuration optimization, and user-centric design are critical for advancing distributed system architectures.

The architecture and design of distributed systems have transformed how we interact with technology, enabling seamless integration of resources across global networks. While the advantages are significant—such as scalability, fault tolerance, and enhanced performance—the challenges, including privacy, scalability, and interoperability, require ongoing research and innovation. As socio-technical systems continue to evolve, their design must focus on adaptability, user-centricity, and sustainable practices to meet the demands of an interconnected world.

REFERENCES

1. White B, Lepreau J, Stoller L, Ricci R, Guruprasad S, Newbold M, et al. An integrated experimental environment for distributed systems and networks. SIGOPS Oper Syst Rev. 2002 Dec 31;36(SI):255–70. Available from: https://doi.org/10.1145/844128.844152

2. Yu J, Wang G, Mu Y. Provably Secure Single Sign-on Scheme in Distributed Systems and Networks. In: 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications [Internet]. Liverpool, United Kingdom: IEEE; 2012 [cited 2024 Oct 30]. p. 271–8. Available from: https://doi.org/10.1109/TrustCom.2012.228

3. Ports DR, Li J, Liu V, Sharma NK, Krishnamurthy A. Designing Distributed Systems Using Approximate Synchrony in Data Center Networks. In: Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI '15) [Internet]; 2015 May 4–6, Usenix, Oakland. USENIX Association; 2015 [cited 2024 Oct 30]. p. 43–57. Available from: https://www.usenix.org/system/files/conference/nsdi15/nsdi15-paper-ports.pdf

4. Yakhno V, Kolumbet V, Halachev P, Khambir V, Ivanenko R. Methods and algorithms of optimization in computer engineering: review and comparative analysis. Data and Metadata. 2024 Jul 17;3:443. Available from: https://doi.org/10.56294/dm2024443

5. Coulouris G, Dollimore J, Kindberg T. Distributed Systems: Concepts and Design. Harlow (England): Pearson Education Limited; 2005.

6. Tanenbaum AS, van Steen M. Distributed Systems: Principles and Paradigms. Upper Saddle River: Pearson Education; 2002.

7. Xi B, Liu Z, Raghavachari M, Xia CH, Zhang L. A smart hill-climbing algorithm for application server configuration. In: Proceedings of the 13th international conference on World Wide Web [Internet]. New York NY USA: ACM; 2004 [cited 2024 Oct 30]. p. 287–96. Available from: https://dl.acm.org/doi/10.1145/988672.988711

8. Saboori A, Jiang G, Chen H. Autotuning Configurations in Distributed Systems for Performance Improvements Using Evolutionary Strategies. In: 2008 The 28th International Conference on Distributed Computing Systems [Internet]. Beijing, China: IEEE; 2008 [cited 2024 Oct 30]. p. 769–76. Available from: https://doi.org/10.1109/ICDCS.2008.11

9. Van Steen M, Pierre G. Replicating for Performance: Case Studies. In: Charron-Bost B, Pedone F, Schiper A, editors. Replication [Internet]. Berlin, Heidelberg: Springer Berlin Heidelberg; 2010 [cited 2024 Oct 30]. p. 73–89. (Lecture Notes in Computer Science; vol. 5959). Available from: https://doi.org/10.1007/978-3-642-11294-2_5

10. Gilbert S, Lynch N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News. 2002 Jun;33(2):51–9. Available from: https://doi.org/10.1145/564585.564601

11. Atzori L, Iera A, Morabito G. The Internet of Things: A survey. Computer Networks. 2010 Oct;54(15):2787–805. Available from: https://doi.org/10.1016/j.comnet.2010.05.010

12. Madden SR, Franklin MJ, Hellerstein JM, Hong W. TinyDB: an acquisitional query processing system for sensor networks. ACM Trans Database Syst. 2005 Mar;30(1):122–73. Available from: https://doi.org/10.1145/1061318.1061322

13. Mottola L, Picco GP. Programming wireless sensor networks: Fundamental concepts and state of the art. ACM Comput Surv. 2011 Apr;43(3):1–51. Available from: https://doi.org/10.1145/1922649.1922656

14. Urdaneta G, Pierre G, Steen MV. A survey of DHT security techniques. ACM Comput Surv. 2011 Jan;43(2):1–49. Available from: https://doi.org/10.1145/1883612.1883615

15. Ananth B. Hybrid Support Vector Machine for Predicting Accuracy of Conflict Flows in Software Defined Networks. Salud, Ciencia y Tecnología. 2024; 4:797.

16. Jain P, Maan V. Enhancing Image Clarity: Feature Selection with Trickster Coyote Optimization in Noisy/Blurry Images. Salud, Ciencia y Tecnología. 2024; 4:1114.

17. Wams JMS, Van Steen M. Unifying user-to-user messaging systems. IEEE Internet Comput. 2004 Mar;8(2):76–82. Available from: https://doi.org/10.1109/MIC.2004.1273489

18. Weiser M. The computer for the 21st century. Scientific American. 1999 Sep;3:94–104. Available from: https://chanwutk.github.io/ucbhciprelim/assets/readings/1991-computer-for-21st-century.pdf

19. Poslad S. Ubiquitous computing: smart devices, environments and interactions. Chichester, U.K: Wiley; 2009.

20. Kreitz G, Niemela F. Spotify—Large Scale, Low Latency, P2P Music-on-Demand Streaming. In: 2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P) [Internet]. Delft, Netherlands: IEEE; 2010 [cited 2024 Oct 30]. p. 1–10. Available from: https://doi.org/10.1109/P2P.2010.5569963

21. Salah T, Jamal Zemerly M, Chan Yeob Yeun, Al-Qutayri M, Al-Hammadi Y. The evolution of distributed systems towards microservices architecture. In: 2016 11th International Conference for Internet Technology and Secured Transactions (ICITST) [Internet]. Barcelona, Spain: IEEE; 2016 [cited 2024 Oct 30]. p. 318–25. Available from: https://doi.org/10.1109/ICITST.2016.7856721

22. Bergstra JA, editor. Handbook of network and system administration. London: Elsevier; 2008. 1016 p.

23. McSherry F, Mironov I. Differentially private recommender systems: Building privacy into the Netflix Prize contenders. In: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining [Internet]. Paris France: ACM; 2009 [cited 2024 Oct 30]. p. 627–36. Available from: https://doi.org/10.1145/1557019.1557090

24. Ramakrishnan H, Keller B, Mirza B, Grama A, Karypis G. Privacy risks in recommender systems. IEEE Internet Comput. 2001; 5:54–62.

25. Amazon Web Services [Internet]. Summary of the Amazon EC2 and Amazon RDS service disruption in the US east region; 2011 Apr 29 [cited 2024 Oct 30]. Available from: http://aws.amazon.com/message/65648/

26. Lewis TG. Network Science: Theory and Applications. 1st ed. New York: John Wiley & Sons, Incorporated; 2011. 524 p.

27. Qin F, Tucek J, Sundaresan J, Zhou Y. Rx: treating bugs as allergies—a safe method to survive software failures. In: Proceedings of the twentieth ACM symposium on Operating systems principles [Internet]. Brighton United Kingdom: ACM; 2005 [cited 2024 Oct 30]. p. 235–48. Available from: https://doi.org/10.1145/1095810.1095833

28. Hunko I, Muliarevych O, Trishchuk R, Zybin S, Halachev P. The role of virtual reality in improving software testing methods and tools. JATIT. 2024 Jun 15; 102(11):4723–4734. Available from: https://www.jatit.org/volumes/Vol102No11/6Vol102No11.pdf

29. Popereshnyak, S., Grinenko, S., Grinenko, O., Kovalenko, O., & Radivilova, T. (2019). Methods for Assessing the Maturity Levels of Software Ecosystems. In Proceedings of the International Workshop on Cyber Hygiene (CybHyg-2019) [Internet]. CEUR-WS; 2019 [cited 2024 Oct 30]. p. 251–61. Available from: https://ceur-ws.org/Vol-2654/paper20.pdf

FUNDINGS

None.

CONFLICT OF INTEREST

None.

AUTHORSHIP CONTRIBUTION

Conceptualization: Yevhenii Tytarchuk, Sergii Pakhomov, Dmytro Beirak, Vasyl Sydorchuk, Svitlana Vasylyuk-Zaitseva.

Writing – original draft: Yevhenii Tytarchuk, Sergii Pakhomov, Dmytro Beirak, Vasyl Sydorchuk, Svitlana Vasylyuk-Zaitseva.

Writhing – proofreading and editing: Yevhenii Tytarchuk, Sergii Pakhomov, Dmytro Beirak, Vasyl Sydorchuk, Svitlana Vasylyuk-Zaitseva.